SHELL FOR THE ILLAC IV

William A. Whitaker
Major USAF

Richard E. Durrett
Captain USAF

Reginald W. Clemens
Captain USAF

TECHNICAL REPORT NO. AFWL-TR-72-33

March 1972

AIR FORCE WEAPONS LABORATORY
Air Force Systems Command
Kirtland Air Force Base
New Mexico

117

R

AFWL-TR-72-33

AIR FORCE WEAPONS LABORATORY
Air Force Systems Command
Kirtland Air Force Base
New Mexico 87117

SHELL FOR THE ILLIAC IV

William A. Whitaker
Major USAF

Richard E. Durrett
Captain USAF

Reginald W. Clemens
Captain USAF

TECHNICAL REPORT NO. AFWL-TR-72-33

FOREWORD

This research was performed under Order Number 17446T and was funded by the Advanced Research Projects Agency (ARPA).

Inclusive dates of research were 1 October 1970 through 31 August 1971. The report was submitted 3 March 1972 by the Air Force Weapons Laboratory Project Officer, Captain Richard E. Durrett (SYT).

This technical report has been reviewed and is approved.

Richard E. Durrett

RICHARD E. DURRETT
Captain, USAF
Project Officer

JAY R. ROLAND
Major, USAF
Chief, Theoretical Branch

William Goodwin for

CARL B. HILLAND
Lt Colonel, USAF
Chief, Technology Division

ABSTRACT

(Distribution Limitation Statement A)

A one-dimensional Lagrangian hydrodynamics computer code (SAP) and a two-dimensional Eulerian hydrodynamics computer code (SHELL) have been successfully written in the GLYPNIR language for the ILLIAC IV. Timing simulations suggest a speed 50 times that of a CDC 6600 for the GLYPNIR SHELL code.

CONTENTS

ILLUSTRATIONS

SECTION I

INTRODUCTION

The Air Force Weapons Laboratory has been engaged in a project with the Advanced Research Projects Agency (ARPA) to provide applications support to the ILLIAC IV project. The Air Force Weapons Laboratory (AFWL), as a representative of a wide variety of users, has undertaken a project to look at the ILLIAC IV from the point of view of the user and assist the project in making the connection to the real world. Large-scale production codes from AFWL are being adapted to the logic, languages, and operating system of the ILLIAC IV. The reprogramming of these codes provides an insight into the usefulness of languages and operating system of the ILLIAC IV, and a basis for timing comparisons on real problems. It will also give an indication of the difficulty in logically modifying real programs to operate on the ILLIAC IV system, and provide an interaction with user programmers to indicate the acceptability of the total ILLIAC IV system to the user in a real world rather than an academic atmosphere. To this purpose AFWL has sent a number of personnel to the University of Illinois to work with the staff there and adapt these programs.

This task has been nontrivial and has involved considerable system and program debugging. The purpose of the present document is to illustrate the method we used to program the SAP and SHELL hydrocodes for execution on the ILLIAC IV. Some basic knowledge of the ILLIAC IV, and associated languages based on the enormous amount of documentation already available on the machine, is assumed. Machine details will appear sketchily where required.

The programming of hydrocodes for the ILLIAC IV has involved several steps. The first of these was to become acquainted with the ILLIAC IV itself, its languages, and the "system" at the University of Illinois. This latter item included such things as learning to use the operating system on the B5500, which was at the University during our first visit, and the B6500 which replaced it. With this in hand it was possible to program a simple code (SAP) just to get practice in the use of the languages and as an exercise in testing the GLYPNIR-ASK-COLLECTOR-LOADER-SIMULATOR route that our later, larger codes would have to traverse. It also meant writing conversion routines to communicate with the

simulator and its Simulated ILLIAC IV Disk files because no one had tried this before and no software existed for our use in this communication.

After these steps were accomplished with SAP, the initial programming of SHELL in GLYPNIR was relatively easy. The initial effort was to write a stripped-down version of SHELL (SHELL62) that would contain the minimum essential elements for an airblast calculation. Extensions could then be made to this basic code to include larger griding and other frills that would not be essential in this first checkout version.

The GLYPNIR SHELL code was tested by using the ILLIAC IV simulator SSK/SSKX on the B5500 and then comparing the results of this run to that of an identical problem run using the standard FORTRAN SHELL on the CDC 6600.

Extension of SHELL62 from this minimal start to the more complete versions was quickly accomplished; however, this job was extremely time consuming on the B5500. The code itself could take half an hour of clock time to compile and another 1-1/2 to 2 hours to assemble. Simulation of a cycle for a grid 5 x 64 would then take another hour or so on the simulator. This then was not an atmosphere in which a real code could be debugged. To reduce the number of debug shots to an absolute minimum, a great deal of extra labor was used.

When the B6500 was installed at the University of Illinois, the expansion of SHELL was resumed. The memory-contained version (called SHELLN) was written and tested. A version was then written to use the disk for expanded storage of the hydro variables. This code, SHELL/OF/THE/FUTURE, can handle up to about two million cells.

SECTION II

SAP

SAP is a simple, one-dimensional Lagrangian hydrocode used for the calculation of nuclear and high-explosive blast waves in air. It is the most elementary of hydrodynamic calculations, but it does exist as a full-production code from which useful data are currently being derived. We have used this code to check out the present state of the ILLIAC IV system and its usefulness for scientific computation. At present, of course, the system is simply a compiler, assembler, and simulator on the University of Illinois B5500 (now B6500). Nevertheless, this was the beginning of the adaptation of much larger and more extensive programs in expectation of the ILLIAC being available.

SAP is particularly useful for this purpose because it has been the standard first-check code for AFWL for a number of different new machines, operating systems, and equipment. In fact, the particular deck we started with for this project was the one used to check out the extended core storage facility for the CDC 6600. For that exercise a program which simulated the extended core in normal core was prepared. This was run for logic verification. Then the simulated reads and writes were replaced by system reads and writes to extended core and the results were checked to ensure that they were identical. Although all the extended core portions of this program were removed, we mention this here because the general technique was the same as we followed in checking out for the ILLIAC IV. The program, as it existed in the particular card deck that we started with, had ample extended core reads and writes, plus such features as an isothermal sound-speed calculation that is included for purely historic reasons and, in fact, is only used to calculate the time step. SAP involves a fair amount of coding which has in the past proven useful for checking out new systems. We emphasize again that SAP is the simplest of all possible hydrocodes, yet historically it has turned out to be extremely useful for checking out new machines. This simplicity, of course, is important because it is easy to discover where any errors might have occurred. For all its simplicity SAP has a sufficient amount of logic and a wide enough distribution of types of arithmetic statements to exercise both the compiler and a fair number of the instructions that are frequently used in any hydrocode.

The general technique used in this and the following code developments has been to first reduce the code to the simplest possible configuration that will run and produce results, and from this build up in steps, to a full-blown code. SAP has an extensive amount of dump and restart capability in it; it calculates standard times, that is, certain selected times are forced to the output so that comparisons may be made between different runs at exactly the same time; and there is a fairly complex analytic equation of state, various options for output and plotting, including tapes and a printer plot, and a fairly elaborate automatic rezone capability. All these were removed, leaving simply a card-read input generation of the problem, raw execution, and printout of the entire output at every cycle. Further, since we were setting up the code to execute on the ILLIAC IV (hopefully in parallel) in the simplest possible configuration, we reduced the number of zones to 64 (actually 62). This, of course, makes absolutely no difference in the FORTRAN version, but conceptually it is satisfying as a starting point for working with the ILLIAC, therefore we actually did make that change. Appendix I gives the SAP code at that point--the so-called stripped, basic version.

It is important to realize that the ILLIAC IV has a number of other features besides simply parallel processing. Perhaps an even more important point is the fact that it is really just a number cruncher as far as some code adaptation is concerned. It does not have the features normally associated with the input/output and data handling of serial computers. One cannot reasonably expect the ILLIAC IV to print output or read cards. Its only source of information from the outside world is through the ILLIAC IV disk (which, we understand, may be changed in the future). This must be initially loaded from the driving computer, that is, a core load and binary data prepared, put on the ILLIAC IV disk, read into the ILLIAC IV core, executed using whatever additional information may be available on the disk, then written entirely to the disk. This disk information can then be postprocessed by the driving computer to provide various visible forms of output. The structure of a code must be made to conform to these limitations..

As it turns out, SAP and a number of other hydrocodes are particularly well suited for this adaptation. Looking at the basic SAP code in appendix I, we see that it starts with an input section that reads cards and sets up the initial mesh. It then goes to a main computation loop and ends in an output or print section. In the more complicated code, of course, the input section

involves the reading of tapes and various restart features and the output section includes writing of tapes, printer plots, etc. Nevertheless, the general structure is still the same. Therefore, we chose to break the code into three parts for convenience and labeled these program I (the input section), program IV (the center computational loop which would be put on the ILLIAC IV), and program II (the output section). By input and output we mean those portions which connect with the outside world, rather than simply to the ILLIAC IV disk. This nomenclature and this way of thinking has been extremely useful in the early adaptation of codes. Thus we have broken up a code into these three constituent parts for running it on the complete ILLIAC system. The first portion, program I (the input section), and the last portion, program II (the output section), would be run on a serial machine, either the driving computer or perhaps some other computer in the ARPA net, while the main computation loop, program IV, would take place on the ILLIAC IV. Communications between these programs would be through the ILLIAC IV disk. This structure is not an unusual procedure because a number of standard hydrocodes already use this method. SHELL, for instance, which is a two-dimensional Eulerian hydrocode, has a very extensive setup section known as CLAM and runs as a separate program. Further, the output processing is also very involved because it is contour plotting for the most part, and is also a separate program, SHPLOT. The normal procedure with SHELL is to prepare a tape with CLAM; SHELL then reads this tape and executes with that data, writing out its results on tape at appropriate times. This output is then postprocessed with SHPLOT and other programs.

After running the basic SAP program in its present form, which is structurally the form we wish to use on the ILLIAC IV, we can then adapt the central section to a language appropriate for the ILLIAC IV and provide the links between this section and the two ends lying in the driving computer.

These two efforts proceed fairly independently and will be described in no particular historical order. Therefore, let us consider the problem of getting the connection between the exterior programs (I and II) and program IV.

There were a number of nontrivial problems associated with this communications task, and apparently very few of these had been faced to any extent by those previously working on the ILLIAC simulator. We were not adapting a program for the ILLIAC IV, but rather for the B5500 simulator (SSK/SSKX) at the University of Illinois, and to a certain extent we had to work within this limitation.

The fact that the B5500 employs a 48-bit word while the ILLIAC IV and its simulator both use a 64-bit word created one of the main problems. Moreover, the general format of the words is entirely different between the two machines. (Note that this will continue to be a problem in the final machine configuration with the B6500 driving the ILLIAC IV.) The first problem, therefore, was to provide a program that would take the initial setup data generated in program I, store it in some common file, then rewrite this data in a form acceptable to the simulator. Later, the reciprocal operation would have to be performed--the output from the simulated ILLIAC IV disk file would have to be transformed back into the B5500 word format for processing by the postprocessor, program II.

Appendix VIII gives a short explanation of the formats of both word types. It is noted that because of the vastly different format of the words, it is important to differentiate between fixed-point and floating-point words when making these translations. In particular, the ILLIAC IV makes the distinction between word formats quite rigorously, while in the B5500 the fixed-point format is just a special case of the floating-point format; a number carried in the FORTRAN program as an integer may appear in floating-point format internal to the machine. For this reason it is recommended that in the future the records being written for the ILLIAC IV core load be entirely in one format--for instance, floating point. Minimally, the fixed- and the floating-point variables should be rigorously separated in the records written or even be put in separate records. Presumably, the same caution holds for alphanumeric or other coded data.*

Finally, an interesting problem related to the number conversion arose that was not recognized until after all of the other features of the code were well checked out and we were able to compare and expect identical (at least to near the 39-bit precision of the B5500) bit patterns from the FORTRAN and GLYPNIR results. When we compared the results in that fashion, they were not identical and after running a number of cycles, actual physical differences were noted.

---

*Although at the time this was written we could not know what the as-yet-to-be-written operating system (OSK) and the ILLIAC control language (ICL) would support in the real ILLIAC system, it is highly desirable that all numbers in a given file should be of the same form. The ILLIAC control language will support the number conversion to and from the ILLIAC disk converting from integer, real, or double presision in the B6500 to integer or real in the ILLIAC; however, all of the numbers in a given file will receive the same conversion.

Now we come to the main part of this section of the report where we will describe the actual programming of the ILLIAC IV part of the code. This was our first effort, so the procedure at this point was extremely cautious. We took the block that we called program IV, the central section of the FORTRAN code with which we started, and reprogrammed it in FORTRAN to adhere to the logic of the ILLIAC IV system. That is, we restructured DO loops where necessary to range from 1 to 64 and isolated those portions of the program which were adaptable for parallel processing. This was essentially a trivial job in the case of this particular code, but the technique is worth pointing out (and it is more interesting in the case of a more involved code).

At this point we were in the position to make an almost card for card translation from the FORTRAN into an ILLIAC higher level language. There were two possible higher lever languages for the ILLIAC available at the time. The first of these, a language having an operating compiler, is called GLYPNIR. The syntax of this language is based on ALGOL, therefore it looks a little strange to FORTRAN programmers. The logic is not all that different from FORTRAN and, after the initial shock wears off, one is able to think in this language very conveniently. The GLYPNIR compiler produced an assembly language program from an initial set of input cards. This program was then put together by the ASK assembler. The other higher language available for the ILLIAC was a specially extended FORTRAN designed for the ILLIAC IV. We conveniently (but improperly) called this *FORTRAN.

This language was designed by the University of Illinois but not thoroughly implemented. The idea was to have short-order implementation of the language (locally known as COCKROACH) that was essentially a *FORTRAN to GLYPNIR translator to gain initial experience with the language. This would eventually be replaced by full FORTRAN compiler which would produce ILLIAC code directly and the GLYPNIR compiler and the assembler would not be used. Thus, being FORTRAN users from time immemorial, we started by transforming the simple FORTRAN code into what we call the *FORTRAN version. The reader attempting to use this procedure will, of course, be familiar in some considerable detail with the syntax of *FORTRAN, so we will not reproduce an explanation of the program at any length. The casual user may wish to glance through this program (shown in appendix II) and quickly peruse the following short explanation. An asterisk in parentheses (*) in the position of a subscript for a FORTRAN variable implies that an entire row across the PEs is associated with this particular variable

and the arithmetic statement there associated should be done simultaneously across the PEs. This is, of course, modified by the mode statement, the mode being a 64-bit word whose bits are set to 1 for those PEs you wish enabled and 0 for those PEs you wish disabled. The *+1 or *-1 constructs simply refer to the quantity in the adjacent PE to the right or left, a quantity derived eventually by routing in the machine. The system is presently limited to 64 in a single variable-implied dimension. It will be extended to take into account variables being dimensioned greater than 64; however, we shall hold this possibility in abeyance for a while because it appears that the overhead in handling that sort of thing in a very general manner might be excessive (or TRANQUIL-like). While handling it in the program explicitly, it does not appear too much of a problem. The program in appendix II gives some idea of the resemblance between this technique and ordinary FORTRAN. Even with all the asterisks the program does look vaguely familiar. However, as we shall later see, this small comfort is not entirely necessary.

At the time this program was written, it was impossible to actually compile the program by machine. In some very vague sense we translated it into GLYPNIR by hand and the result of this translation is shown in appendix III. Two or three things should be pointed out here. First, the overall ALGOL syntax of the program is a little strange to the FORTRAN-adapted eye, but most of the arithmetic statements and the bulk of the logic does transform almost one for one from the FORTRAN. There are some major differences which have to do with the declaration of variables at the beginning of the code, i.e., the FORTRAN COMMON structure. In this language, all of the subroutines that we call must be placed previous to their first use. Thus the lowest order subroutines appear first and the main program last.

We have here added to the program those portions which couple it to the ILLIAC IV disk, the simulated read and write statements. One of the features that we should bring up at this point, which has since been changed somewhat but was necessary in the original version of this code, was that all input had to be done with a single long-read statement from the ILLIAC IV disk. Thus all information came in in the form of words written across PE memory. This was perfectly appropriate for things like pressure, velocities, and other zone quantities, but it is less useful for those quantities in an ordinary code that represent single variables rather than dimensioned variables. These variables will be identified as CU variables in the ILLIAC and must be read in as elements

of a PE variable and then set into their respective CU variables. For this purpose we used what is known as the Z block. A variable Z is dimensioned as necessary for the total number of variables we wish to bring in. All the appropriate CU variables in program I are equivalenced into the Z block that is normally used as the first dimensioned variable in the program. One must then perform an action in the GLYPNIR program equivalent to the equivalence statement in program I. Thus, at the beginning of program IV the individual numbers are pulled out of the Z block and put into CU variables with a GRABONE statement. Of course, before every write on the disk the appropriate CU variables must be placed back into the PE variable Z. (NOTE: This is another case where floating- and fixed-point numbers must be handled somewhat differently and reemphasizes the usefulness of either separating the variables or keeping them all floating point. Our codes have tended to use the latter technique.) Strangely enough, the Z-block technique and terminology are not unique to this code system or machine. It is taken directly from the standard SHELL procedure to arrive at its present form. Other than these items, the GLYPNIR program is quite straightforward and very closely resembles the FORTRAN program. It is this that makes us say that the general structure of the GLYPNIR is in fact very FORTRAN-like, although disguised behind the ALGOL syntax.

A number of individual, very special problems turned up during the checkout of this code. Some misunderstandings and difficulties in the compiler were straightened out and these need be of no concern to the reader because they have been changed. There are, however, a couple of points which might be mentioned as still being of some use. The first and foremost of these is the use of mode control with the ILLIAC because this is something that the programmer will have to get used to. It is somewhat tricky and, in general, we have tended to play safe in certain instances by just setting the mode TRUE (i.e., all 1s) on all occasions for which we do not have a very specific reason for setting it to something else. Routes, for instance, can occur under mode control, but a statement routing a row of numbers under the current mode may leave several PEs out in the cold when, in fact, you wanted values brought up from them and routed. Secondly, we note that the ALGOL IF ... THEN ... ELSE statement is perhaps somewhat superflous in the context of the parallel machine in that its main advantage in ALGOL is to provide simply two different routes, only one of which is executed during single flow through the procedure. In a serial machine this obviously is the desired technique. In a parallel machine, however, we normally

will have a number of processors operating for one branch and others operating on the other branch. Moreover, it is found that this particular GLYPNIR construct generates a great deal more actual code than just setting the ELSE clause initially for all PEs and then following this with an IF ... THEN statement. This is a small point, but it explains why the more obvious uses of the IF ... THEN ... ELSE statement was not employed in this particular code.

This deck then, combined with programs I and II shown in appendix IV, represents the complete package that was simulated. The code produced the results expected on comparison with the original FORTRAN program.

One should say something about the actual physical operation. These remarks are transitory, of course, and will change as the operation changes. By the time this report is published, the B6500 at the University of Illinois will have an entirely different operating system. Nevertheless, the following difficulties may still be encountered. First, the B5500 expects a BCL character set (the B6500 accepts EBCDIC or BCL) and this, of course, will make certain transformations a little more difficult to those of us with only 026 key punches. This can be programmed ahead of time at one's local installation and transformed character by character into an acceptable deck. There is one rather unfortunate feature in the Burroughs system, however--the system will stop the card reader if an illegal character is encountered during a read. This is particularly upsetting in situations where the error occurs in columns that the program or compiler will later ignore. We arrived with several decks that had illegal multiple punches in columns 73 through 80. Card by card by hand with a key puncn we had to correct every one. It would be a happier situation if the system ignored illegal punches altogether and treated them as blanks other than in column 1 (which signifies a control card), but this is not the case.

The system does not accept serial batch runs, so the three individual programs had to be loaded separately, one following the completion of the other, otherwise the system would attempt to execute a program before its predecessor had been completed and find no data to work on. Further, the printer unloaded regularly, so the compilation and the execution listing need not be together or even in order. Certain limited types of machine failures were common during the operation and it was found advisable to be able to restart the simulator where the machine dropped off because the simulations themselves often ran for an hour or so. The program that performed this task is shown in appendix V. Here, we pick up the last completed cycle of our calculation from our output

file, have the capability to make changes as desired to the Z block, and write a new input file to pick up the simulation with. It is noted that the more straightforward use of the restart feature of SSK/SSKX was not possible at the time we ran these simulations because it lost track of its disk files during a restart.

The extension of SAP from a straight 64 (or, in fact, 62 zone) mesh to one of arbitrary width was straightforward and perhaps fairly simpleminded. Nevertheless, it works well. We view the mesh as a number of 64-word blocks, overlapping one zone at each end with the next block, as shown in appendix VI. Due to the first order differencing scheme used in SAP, we must have information from the previous cycle about the neighbors, left and right, of any cell before we can update it in the mesh. Thus, to update zone 62 in this first order code, one must have information from zones 61 and 63, both of which are available in the first row. One cannot update zone 63 with the information available in the first row. We have information about zone 62, but we do not have information about zone 64. Zone 63 (the last zone of the first row since the present nomenclature has been fixed with PEs running from 0 to 63), the zone associated with PE 63, must therefore be repeated in the second row so that it can be updated, but as noted we will need one zone to the left of it. Thus the information from PE 62 of the first row must be loaded into PE 0 of the second row, PE 63 into PE 1, and the second row continues until the last two PEs are reached. These last two PEs must overlap the first two PEs in the third row, and this continues for as many rows as are needed. One must point out that while this overlap must occur during the run, the actual movement of the numbers does not occur until after a cycle has been completed. That is, we must use the old value of the quantity in the 62nd zone in PE 0 of the second row to update the 63rd zone in PE 1 of the second row. Of course, one can apply the same sort of thing or essentially create a zone 0 for the center boundary condition in this spherically symmetric code. These arrangements and scheduling of zones must be made in program I and taken out in program II before the final upward processing.

This initial storage scheme with the data in the PEs just described made the loading of these end PEs run at extremely low efficiency because the data from PE 62 of each row had to be moved to PE 0 of the next row. Similarly, data from PE 1 of each row had to be moved back to PE 63 of the previous row. Thus, one, or two at the most, PEs could be active during this adjustment phase, making it much less efficient than the actual calculation where 62 or 64 PEs were active at a time.

This was a very crude way of handling data and a much more sophisticated method immediately came to mind. For instance, one can rotate the second row two PEs to the left with respect to the first, the third with respect to the second, by an additional two PEs, etc. In this way one can make the entire readjustment process in parallel with all PEs active, rather than making a separate adjustment for each row. This is perhaps a trivial additional complication but one which we did not wish to get into the first time through. SAP, although a simple code, has many of the features of the more complex codes, and because of its simplicity, it allowed major changes in programming and storage with only a minimal rewriting of code. The coding of SAP brought out many of the difficulties of the present operating system and allowed us to exercise solutions thereto. It also provided a great deal of experience in actual physical manipulation of codes in the machine.

This version of SAP is certainly not the most general code even for its particular limitations to one-dimensional spherical hydro. The main production codes are normally run with, for instance, complex equations of state. The equations of state normally used in this code are interchangeable with other codes (like SHELL) and were written as a part of that effort. Some of the more elaborate rezone techniques have not been included in this version. Because we believe these are really details and do not further illustrate our main purpose in this initial run, we did not include any of the fancier additions such as radiation, complex boundary conditions, etc. We did not face some of the restart problems because they will be very specifically related to the driving machine and necessary details were not available for our investigation. Many of these problems will be taken care of in later versions of the code and perhaps one of these later versions of SAP will be brought up to full strength immediately before the full availability of the ILLIAC IV itself.

Finally, for comparison purposes the ASK code assembly language which was produced by GLYPNIR was compared with an assembly language program that was written from the same groundwork, but hand-coded, rather than using the GLYPNIR compiler. This might conceivably be looked upon as what a better compiler should be able to produce. It was noted that this hand-compilation gave a fair reduction in number of instructions generated (a factor of 3), but that the improvement was not especially startling in execution time. One can say that for a first cut the GLYPNIR compiler generates quite respectable code.

The pure assembly language program is listed in appendix VII and was compiled and executed as a part of the three-program operation. It gives identical results to the other procedures.

SECTION III

SHELL

1. THE METHOD

With the learning experience of SAP behind us, we initiated an effort to convert the much more extensive SHELL hydrocode to the ILLIAC.

This effort proceeded along two lines. On the one hand, Major Whitaker and Mr. Needham took a FORTRAN SHELL listing and in the spirit of the SAP effort, wrote a SHELL version from it. This then could be held until the COCKROACH translator was available. This also could be used as a stepping stone in the translation of the serial FORTRAN logic to parallel GLYPNIR logic. This effort produced a nearly complete *SHELL deck for rows up to 62 zones wide. This code was later run through the *FORTRAN translator but was not compiled or executed.

2. FORTRAN SHELL

One is appalled by the large amount of code in a working FORTRAN version of SHELL. This is traceable to the limitations of the various machines and compilers that SHELL has run on. Once added to SHELL, this code has never been removed. Indeed, SHELL works as it stands. In any removal process the most careful attention to detail would be required to ensure that necessary and currently useful data are not deleted inadvertently.

Leaving everything that has accumulated over the years in the code is quite acceptable when moving the code from one FORTRAN compiler to another because extensive time and effort at a keypunch would be needed to remove those parts of the code which are of questionable use. Only a few minor changes are usually necessary to get the entire code running on a new compiler. This is hardly the case in reprogramming the code for a new language (GLYPNIR) and a new machine architecture (the ILLIAC IV). Since the code had to be rewritten from scratch and carefully debugged, this was the most reasonable time to delete much of the overhead from SHELL and make the programming task in GLYPNIR at least smaller.

In rewriting the code, it was decided to remove the capability (such as it was) to handle vacuum zones because this only gives stopgap answers to the questions one tries to solve by using this technique. The artificial viscosities

were also removed from PH1 of the code because these have not been used in the FORTRAN version of SHELL in many years. The viscosity calculations almost double the number of lines of code in PH1.

The INPUT and EDIT routines were moved to the driving computer programs I and II and with the deletions the length of SHELL was reduced from about 2500 to 4000 cards in its various FORTRAN versions to approximately 500 to 700 cards in GLYPNIR. Thus substantial deletions to PH1 and PH2 (artificial viscosities, vacuum zones) dictated that we start from scratch. If we had tried to adapt the code as it stood, the known inconsistencies in the code that handles variable zoning in the mesh could have caused trouble. In this stripped-down version, these inconsistencies could be corrected easily.

Our approach was first to write a stripped-down version of the code and work up from there. We decided to rederive the basic difference equations on which SHELL is based. This rederivation crystallized our thoughts and helped to bridge the gap between the serial logic in the FORTRAN version of the code and the actual equations we were trying to solve. This approach seems highly desirable for many large codes being transferred to the ILLIAC IV.

At this stage the problems to be encountered by a computer translation of a serial code into a parallel code (or indeed the translation of a large code by someone who is not familiar with it) became only too apparent. The programmer who is familiar with the physics of the code could make a number of changes, deletions, etc., but the programmer trying to make a strict one-for-one translation of the code would be forced into a number of costly inefficiencies if he wished to preserve the code. What is basic to SHELL is perhaps a half dozen lines of physics presented in the next section and not the thousands of cards of superstructure into which they are embedded so as to apply them to a problem.

Now we will briefly sketch the rederivation of the SHELL difference equations as made by Captain Durrett. Except for the notation, which is now much clearer, there are no changes in the basic equations from those that were derived by others at earlier times. There are differences in the boundary conditions derived and in the update to the total energy in the system (ETH) that were discovered in this rederivation. Earlier authors had made those cosmetic changes necessary to convert the constant zoning results to variable zoning results, but had missed several dimensionless multipliers which would have been included if a consistent rederivation had been done with variable

zoning in it from the start. Thus it may be that the poor relative error that SHELL has reported over the years while using a variably zoned mesh is really an error in the computation and reporting rather than an actual error in the calculation.

3. THE SHELL DIFFERENCE EQUATIONS

The basic hydrodynamic equations integrated by SHELL are

$$\left(\frac{\partial}{\partial t} + \vec{u}.\nabla\right) \rho + \rho\, \nabla.\vec{u} = 0 \tag{1}$$

$$\rho \left(\frac{\partial}{\partial t} + \vec{u}.\nabla\right) \vec{u} + \nabla p = 0 \tag{2}$$

$$\rho \left(\frac{\partial}{\partial t} + \vec{u}.\nabla\right) E + \nabla\cdot p\vec{u} = 0 \tag{3}$$

$$p = p\,(E,\rho) \tag{4}$$

Taking equations (2) and (3) and ignoring the $(\vec{u}\cdot\nabla)$ operator, we get

$$\rho \frac{\partial \vec{u}}{\partial t} + \nabla p = 0 \tag{5}$$

$$\rho \frac{\partial E}{\partial t} + \nabla\cdot p\vec{u} = 0. \tag{6}$$

Equation (5) gives us one of our difference equations directly. Equation (6) must be manipulated somewhat before it can be used. The E in equation (6) is

$$E = E_m + 0.5\,(\vec{u}\cdot\vec{u})$$

i.e., specific material plus specific kinetic energy. Therefore,

$$\frac{\partial E}{\partial t} = \frac{\partial E_m}{\partial t} + \frac{\partial \vec{u}}{\partial t} \cdot \vec{u}$$

which, using equation (5), can be written

$$\frac{\partial E}{\partial t} = \frac{\partial E_m}{\partial t} - \left(\frac{\nabla p}{\rho}\right)\cdot \vec{u} \tag{7}$$

We now expand the divergence in equation (6) to get

$$\frac{\partial E}{\partial t} + \frac{1}{\rho} \left(p\nabla.\vec{u} + \vec{u} \;.\; \nabla p\right) = 0$$

and then use equation (7) to cbtain

$$\frac{\partial E_m}{\partial t} - \left(\frac{\nabla p}{\rho}\right).\; \vec{u} + \frac{p\nabla.\vec{u}}{\rho} + \frac{\vec{u}.\nabla p}{\rho} = 0$$

or canceling terms

$$\frac{\partial E_m}{\partial t} + \frac{p\nabla.\vec{u}}{\rho} = 0 \tag{8}$$

This can be expanded in two-dimensional cylindrical coordinates to

$$\frac{\partial E_m}{\partial t} = - \frac{p}{\rho}\left(\frac{1}{r} \; \frac{\partial ur}{\partial r} + \frac{\partial v}{\partial z}\right) \tag{8a}$$

This then is the second equation we want to solve.

The finite differencing of equations (5) and (8a) is straightforward. Equation (5) becomes, in cylindrical coordinates and the nomenclature of figure 1,

$$\frac{\tilde{u}_k - u_k^{(n)}}{\Delta t} = - \frac{1}{\rho_k} \left(\frac{p_i - p_{i-1}}{r_i - r_{i-1}}\right) \tag{9}$$

$$\frac{\tilde{v}_k - v_k^{(n)}}{\Delta t} = - \frac{1}{\rho_k} \left(\frac{p_j - p_{j-1}}{z_j - z_{j-1}}\right) \tag{10}$$

Equation (8a) becomes

$$\frac{\tilde{E}_m - E_m^{(n)}}{\Delta t} = - \frac{p_k}{\rho_k}\left(\frac{1}{r_k} \; \frac{\bar{u}_i r_i - \bar{u}_{i-1} r_{i-1}}{r_i - r_{i-1}} + \frac{v_j - v_{j-1}}{z_j - z_{j-1}}\right) \tag{11}$$

where

$$\bar{u} = \frac{u^{(n)} + \tilde{u}}{2}$$

$$\bar{v} = \frac{v^{(n)} + \tilde{v}}{2}$$

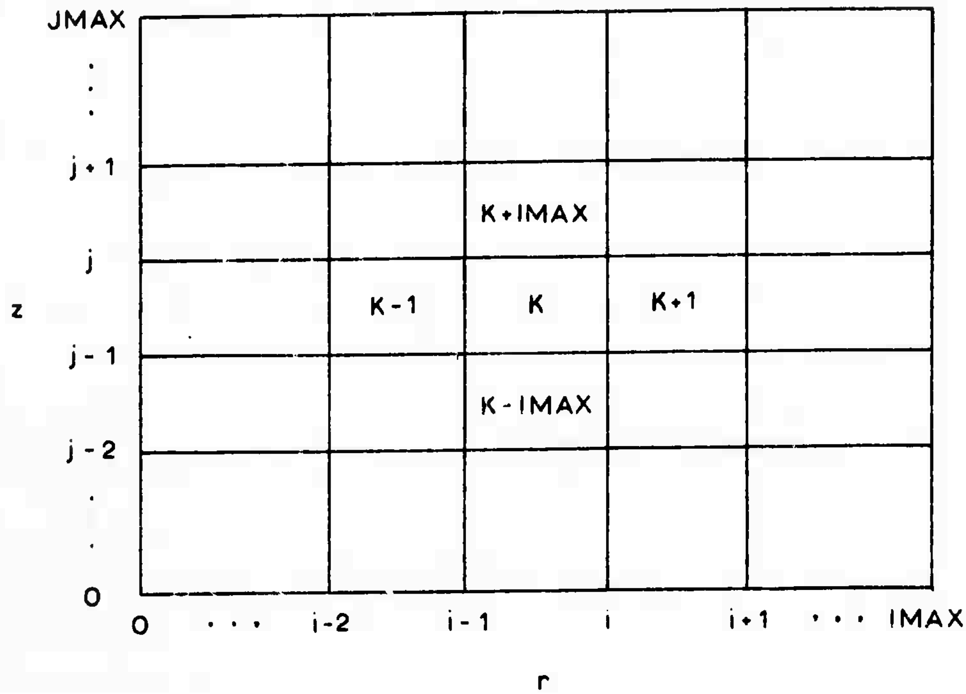


Figure 1. The SHELL grid

The $\tilde{u}$, $\tilde{v}$, and $\tilde{E}_n$ are the new velocities and internal energy at time (n+1) based on the pressure effects of equations (2) and (3) only before accounting for any mass motion.

The distance r_i refers to the right edge of the cell k while r_k refers to the center of the cell, i.e.,

$$r_k = \frac{r_i + r_{i-1}}{2}$$

Similarly, the p_i, u_i, and v_i are defined at cell boundaries while u_k, v_k, p_k, ρ_k, and E_m are defined at cell centers. Thus, for constant zone size p_i, for

example, is defined by

$$p_i = \frac{p_k + p_{k+1}}{2}$$

In the case of nonconstant zone size this becomes

$$p_i = \frac{p_k \Delta r_{k+1} + p_{k+1} \Delta r_k}{\Delta r_{k+1} + \Delta r_k}$$

where

$$\Delta r_k = r_i - r_{i-1}$$

Now this choice of $\bar{u}$ and $\bar{v}$ is used in the energy equation to achieve energy conservation; however, in the case of variable zone size the conservation is no longer assured. SHELL for the ILLIAC was programmed to allow for variable zone size.

Equations (9), (10), and (11) are solved in and constitute what is called phase 1 of the calculation. Equation (1) is solved in what is called phase 2, as follows.

Rewriting equation (1)

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \vec{u} = 0$$

or in two-dimensional cylindrical coordinates

$$\frac{\partial \rho}{\partial t} = -\left(\frac{1}{r}\frac{\partial \rho u}{\partial r} + \frac{\partial \rho v}{\partial Z}\right)$$

The finite differencing of this equation gives

$$\frac{\rho_k^{(n+1)} - \rho_k^{(n)}}{\Delta t} = -\left(\frac{1}{r_k}\frac{\rho_i u_i - \rho_{i-1} u_{i-1}}{r_i - r_{i-1}} + \frac{\rho_j v_j - \rho_{j-1} v_{j-1}}{z_j - z_{j-1}}\right) \tag{12}$$

SHELL uses this formulation except that the ρ_i and ρ_j are taken not on the boundary as indicated in equation (12), but rather the density chosen is that of the center of the cell donating the mass. Also, the velocity chosen is not precisely u_i or v_j as indicated, but rather a second order choice is made. This is done to improve the behavior of the method in areas behind a strong shock. The velocity chosen is called the "rear of slug velocity" and is shown in figure 2. The volocity u_i' is chosen as a linear interpolation between u_k and u_{k+1}, to the point r such that with velocity u_i' the rear of the slug will, in time Δt, move exactly to position r_i, the boundary. The amount of mass between positions r_i' and r_i will therefore move into cell k+1 in the current time step (at a density of ρ_k). With these modifications equation (12) is solved in phase 2.

With an appropriate equation of state this form of SHELL was coded for the ILLIAC IV.

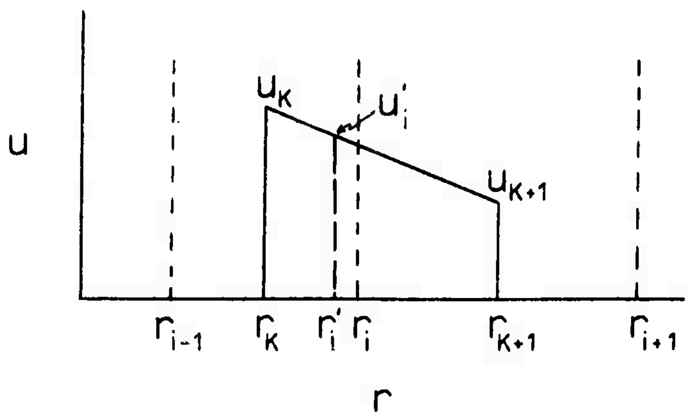


Figure 2. Rear of Slug Velocity

4. SHELL62

To program the equations above, it was first necessary to decide on a method of data storage as the entire algorithm for solution is tied to this choice. If one choses a computational technique, it is necessary that the correct data be in the appropriate PEM (processing element memory) or nearby for efficient use of ILLIAC IV. Alternately, given a choice of storage scheme, some algorithms will be much more efficient than others.

The scheme that first comes to mind for SHELL is a contiguous mapping of PEs across a row of the SHELL mesh as shown in figure 3. With this scheme one can then obviously store all the required information for the first column in

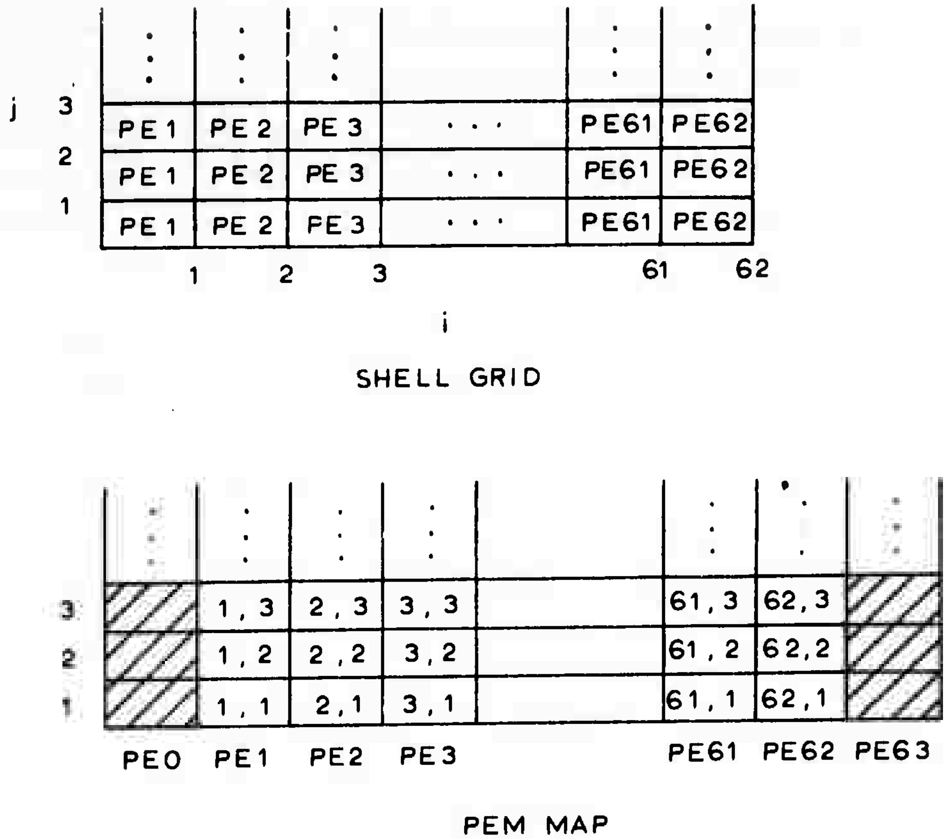


Figure 3. SHELL62 Mapping of Cells and PEs

PE 1, for the second column in PE 2, etc. Because of the nature of the SHELL equations and the explicit time differencing scheme used, the future values of hydrodynamic variables for any cell are determined from values in the cell and its four neighbors to the right and left and above and below only. These data are conveniently available from the right and left by routing instructions in GLYPNIR and above and below from different sections of the corresponding cell PEM. For the initial implementation of SHELL we chose to limit the width of the mesh such that one row of the problem could be contained across the set of PEs. Our experience with SAP showed that it would then be quite easy to extend the code to arbitrary width. However, as became apparent in coding SAP, including this extension in the original code would have added an unnecessary load of detail (mostly in keeping track of the mode) to the coding effort at a time when attention should have been paid to more significant problems.

As it was, writing SHELL first to only 62 columns wide (SHELL62), then extending it to arbitrary width (SHELLN), we discovered only two or three errors (all in handling the mode) in the first code (SHELL62). No additional errors

were found in the extension to arbitrary width. To handle the boundary conditions conveniently (and simplistically), we required that the row width be limited to 62 cells or less in this version. PE 0 and PE 63 (or the last active cell +1) are then used to store values which will give the appropriate boundary conditions when the row is calculated. To this end we chose to insert precalculated values in these PEs.

The implementation of the SHELL code in GLYPNIR for 62 zone width is shown in appendix IX. A description of the general features of the code follows. This was intended to be a code complete enough to execute on the ILLIAC IV simulator on the B5500 computer, and this was in fact accomplished successfully in January 1971.

We have no intention of expanding the code on a line by line bases here because we feel that little could be gained by this exercise. But we do feel that a certain number of comments is in order.

The FORTRAN version of SHELL proceeds through the calculational mesh by columns--bottom to top, then left to right. The GLYPNIR version processes the mesh by rows--bottom to top (left to right in SHELLN). This is a minor change, but it makes the code much easier to handle in parellel. Because FORTRAN SHELL updates its arrays in on top of themselves, some of the unupdated quantities must be saved for use in updating the next cell. This logic in the code has been extensively changed by processing a row of cells at a time; in some cases a whole row must be saved and in others none at all. Further, certain geometrical constructs that involve X and DX as well as the mode are calculated once in SETDXDYETC rather than continually calculating them throughout the code.

Another interesting point can be noted in ES, the air equation of state used in this code. In the FORTRAN version of the code the arguments of the exponentials are extensively checked, basically for speed. By checking the argument, we can often set the result to 0 or 1 and avoid the loss of time spent in calculating the exponential. Since the equation of state is called at least once per zone per cycle, even a small saving in time is worthwhile. In the GLYPNIR version, however, the shoe seems to be on the other foot. Here we can expect that of the 64 zones being calculated at once, one will require us to calculate an exponential. Since this is the case, it takes no longer to calculate 64 exponentials than it does to calculate one, so we have eliminated the argument checking as excess overhead.

The subroutine ERROR references the file "LINE." This is a construct of the simulator, not of the ILLIAC itself and is our only deviation from the use of constructs that will be available in the real ILLIAC system. (Of course, with the coming of the real operating system for the ILLIAC, the SIMREADs and SIMWRITEs will become simply READs and WRITEs.) This is used because communication with OS4, the ILLIAC operating system, had not been defined or implemented at the time this code was written. When this implementation is complete, the SIMULATE (LINE, -) will be replaced by an appropriate call to the system to return our error code. (See SHELL/OF/THE/FUTURE listing in appendix XI.) As noted before, certain constructs that are familiar to the FORTRAN programmer either have not been implemented in GLYPNIR or they exist in a rather primitive form. Various things can be cited that fall into this category, such as the partial implementation of the FORTRAN EQUIVALENCE facility and the rather primitive I/O facility.

None of these lacks is really serious because each can be programmed around. The I/O is probably least serious because we will probably want to do unbuffered I/O with our large data arrays. The ability to equivalence CU variables into PE variables would greatly simplify our INPUT and OUTPUT routines. As it is, we merely use replacement statements and the GRABONE and PUTCPR functions to pack and unpack our PREAL variable Z. Our only use of the equivalence facility as it currently exists is found as the seventh line of the subroutine ES, where we equivalence RHO and CS.

One may now look at the code as a whole and question the interaction between the storage scheme and the code of the routine. Surely the storage that we are using, storing the grid by rows, is the simplest conceptually that we can hope to achieve (figure 3), so it is of interest to discuss the reasons that we, or others, would go to some other method of storage.

In other types of calculations one is given an array of some predetermined size to operate on, and it is desirable to be able to come close to this size in one storage scheme. For instance, in the code just described with storage by rows, a grid 62 zones wide (or n x 64 - 2, in general) would be most efficient in terms of storage and PE usage. If we were given a mesh instead of being able to choose one, we could run into serious efficiency problems. We would want to choose a more compact storage scheme (say 8 x 8 rows) and reduce this waste. We do not have this problem with hydrocodes because we will be working on meshes considerably larger than 64 in a given direction and we can

always use the extra rows in a given direction to secure more resolution. The reasons for considering such storage would be with two ends in view. First, the least efficient part of the SHELL62 code comes in presetting the boundary conditions at the axis and the right of the grid in preparation for the actual hydro calculation. This is done with essentially only one PE turned on. A more complicated storage scheme (say 8 x 8 or C-skew) would allow us to process more than one boundary condition at a time because the boundary cells would not all be in the same PE. This would increase the efficiency of that part of the code. For instance, using 8 x 8s, we could process eight cells at a time. With a 7 x 9 C-skew we could effect 100 percent PE efficiency by updating 64 boundary conditions at a time.

This more compact, blocked storage would also help in calculations of ES and opacities of materials because one could expect a compact region would have essentially the same conditions throughout. Hence, the number of iterations for each cell would be approximately the same or the part of a table searched would be small.

For these reasons we investigated reprogramming part of SHELL using 7 x 9 C-skew storage. Although the boundary conditions would be performed at higher efficiency, the overhead in doing just the normal hydro calculations would more than outweigh the advantages.

Time was not available to investigate the less than optimal 8 x 8 case. Nor was there time to investigate the possibility of transferring the arrays before and after certain operations. These are not currently thought to hold any probability of a big payoff, but certainly should be considered in the future.

5. SHELLN

SHELLN is an expansion of SHELL62 to handle rows of any width. The principle difference in the codes is in the handling of the row storage. Figure 4 shows the mapping of the SHELL grid onto the processing element memories. As in SHELL62, one dummy cell is required at each end of the row to handle the prestored boundary conditions. Because we are free to choose the number of cells in a row in a typical hydro calculation, we will arrange to have only two cells per problem row that do not represent hydro cells.

SHELLN was written to handle a problem of any size provided it could be memory contained. A listing of the code is given in appendix X. The code shown compiled on the B6500 at the University of Illinois in approximately 12

SHELLN

j										
		⋮	⋮	⋮		⋮	⋮	⋮	⋮	
	3	PE1	PE2	PE3	. . .	PE62	PE63	PE0	PE1	. . .
	2	PE1	PE2	PE3	. . .	PE62	PE63	PE0	PE1	. . .
	1	PE1	PE2	PE3	. . .	PE62	PE63	PE0	PE1	. . .
		1	2	3		62	63	64	65	

i

ROW 3	9	128,3	129,3	130,3				
	8	64,3	65,3	66,3	67,3	. . .	126,3	127,3
	7		1,3	2,3	3,3	. . .	62,3	63,3
ROW 2	6	128,2	129,2	130,2				
	5	64,2	65,2	66,2	67,2	. . .	126,2	127,2
	4		1,2	2,2	3,2	. . .	62,2	63,2
ROW 1	3	128,1	129,1	130,1				
	2	64,1	65,1	66,1	67,1	. . .	126,1	127,1
	1		1,1	2,1	3,1	. . .	62,1	63,1
		PE0	PE1	PE2	PE3		PE62	PE63

Figure 4. SHELLN Mapping of Cells and PEs

minutes of processor time (30 minutes real time but with other users in the mix). One cycle was then executed on the ILLIAC IV simulator for a problem mesh 5 rows tall and 70 columns wide in 30 minutes of processor time (60 minutes real time with no one else in the mix). The answers agree precisely with those obtained from the FORTRAN version of SHELL. One may notice that PH3 and REZONE are included in the code; these sections compiled but were not executed because of time limitations.

Figure 5 shows a sample line of code from SHELL62 and SHELLN demonstrating the principle difference between them and the solution of a problem unique to parallel processing. The line shown in figure 5 is quite relevant to the code and constitutes the computation of the acceleration of the fluid in the radial direction for an entire row of the problem. This is the finite difference form of the radial component of one of the three partial differential equations that SHELL seeks to solve. A detailed explanation of the code line follows.

MODE is a boolean variable (single word) that specifies bit by bit which PEs are on and which are off for most computational purposes. For a SHELL62 mesh as shown in figure 3, the appropriate MODE pattern would be 011 ... 110; that is, the first (zeroth) and last (63rd) PEs are off and all the rest are on. This boolean value is stored in MI and used throughout the code.

The variable U is a PREAD vector dimensioned to 100; that is, 101 storage locations are reserved for U in each PEM (processing element memory), and it is type REAL (as opposed to INTEGER or BOOLEAN). The variable K is type CINT, which means it has a single integer value. The construct U[K] then refers to the (K+1)th value (because we start counting from zero, naturally) of U, which in general will be different in each PEM. TAU is an area term and has a different value in each PEM. DY is single valued (for each J which is also single valued) and represents cell height.

We now come to an interesting construct of GLYPNIR, that of a route. As shown in figure 5, it is a route to the right, an amount 1, of variable PR, the pickup being under a mode pattern shifted 1 to the left from MI. To see what happens, first shift the mode one bit to the left (end around). The pattern will then be 11 ... 100. Under this mode pick up variable PR. This pickup will be accomplished in PE 0 through PE 61 and nothing will be done in PE 62 and PE 63. Next, route (routing registers are always on regardless of the mode) the value picked up one element to the right. The mode is returned to its

```
SHELL62:

    MODE=MI

    U[K]=U[K]+TAU*DY[J]*(RTR(1,,PR)-PR)*DT/(DX*AMX[K]);


    WITH TYPES:

        PREAL VECTOR U, AMX[100];

        CINT K, J;

        PREAL TAU, PR, DX;

        CREAL  VECTOR DY[100];

        CREAL DT;


SHELLN:

    LOOP I=0,1,IBLK DO BEGIN

        MODE=IMODE[I];

        U[K]=U[K]+TAU[I]*DY[J]*(RTR(1,,PR[I+IR])-PR[I])*DT/
             (DX[I]*AMX[K]);


        END;

    Where IR is type PINT and = 0,0,0,...,-1
```

Figure 5. Sample Line of GLYPNIR Code

original value (MI) and the routed values are available for computation in the PEs to which they have been moved. Now since PR is type PREAL, there can be a different value in each PEM. As can be seen in appendix X, a pressure term is actually stored in PR so that when PR is subtracted from RTR (1,,PR) we actually take the difference in pressure between each cell and its neighbor to the left. Of particular concern is the value of PR in PE 0. Although this PEM does not represent an actual hydro cell, a value of PR must be available for the route. An appropriate boundary value is stored in PEO before this line of code is reached. The remaining variables in the line represent the time step, cell width, and cell mass.

Figure 5 also shows the same computation for SHELLN and the changes required to accomplish it for any row width. The single statement is replaced with a loop over the number of PEM rows required to store one row of the actual hydro mesh. We now have an appropriate mode, stored in IMODE, for each PEM row. The value for IMODE[0] would be 011 ... 1 for the mesh of figure 4, while IMODE[1] would be all on and IMODE[2] would be 11100 ... 0. For I = 0, the line of SHELLN code executes similar to that for SHELL62 above. Now since the area and cell width are different, in general, for each cell, TAU and DX have been expanded to vectors.

These usages are simple and straightforward. The significant development comes in the handling of PR. First of all, it is expanded to a vector like TAU and DX and then, to do the end around route correctly (that is, to connect cell 64,1 with 63,1 (and not 127,1)), it is necessary to subtract 1 from the subscript on PR in PE 63 only. This is accomplished by adding IR, which is equal to 0,0 ..., 0, -1, to I after the mode is shifted end around one to the left, similar to that described for SHELL62 above. This construct simply and effectively connects the three PE rows to form one problem row for all routes to the right. For routes to the left, a similar construct, employing IL = 1,0,0,...,0 in place of IR, is used.

6. SHELL/OF/THE/FUTURE

Although SHELLN completed the development of a code similar to the regular production SHELL code used at AFWL, the fact that it is memory contained restricts the problem size to about 20,000 cells, the same number that can be run on our CDC 6600. We therefore expanded the code once more to use the ILLIAC IV disk for problem storage. The result was SHELL/OF/THE/FUTURE, which

solves the problem by blocks,reading and writing them to disk in a fashion similar to the use of extended core at AFWL. A listing of this code is given in appendix XI. Programs to write and read the ILLIAC IV disk files are given in appendixes XII and XIII. These correspond to the programs I and II for SAP discussed earlier.

SHELL/OF/THE/FUTURE will be able to use the full capabilities of the ILLIAC IV in solving a two-dimensional hydrodynamics problem with a single material. However, since this code was developed, we have written a new hydrodynamics code at AFWL that we will probably want to code in GLYPNIR and use on the ILLIAC IV in place of SHELL/OF/THE/FUTURE. This new code will include the ability to handle any number of materials as well as containing other features, and it will employ new differencing techniques.

SECTION IV

CONCLUSIONS

The basic purpose of this exercise was to translate working one- and two-dimensional hydrodynamic computer programs from serial logic applicable to a machine such as the CDC 6600 to parallel logic appropriate for the ILLIAC IV. To accomplish this we found it most convenient to reduce the problem to its basic elements and start from the beginning. Even with the more adventuresome effort, the SHELL code, we were able to make rapid progress after rederiving the basic difference equations.

We expected to encounter difficulties in using the ALGOL-like GLYPNIR language but actually found after working with it for several weeks that we were able to think in the language with relative ease. The real difficulties that we found in the code translation effort were all related to the parallel structure of the ILLIAC IV.

The SHELL code as it exists at AFWL consists of a few simple routines which solve the hydrodynamic equations coupled with a fairly large number of support routines which make a working and useful code. Included in this group are the massless (for some applications massive) particle transport routine generally used to follow Lagrangian interfaces and an automatic rezone routine. For problems with several materials a diffusion limiter is available.

The coding in GLYPNIR of the basic hydrodynamic equations was relatively straightforward. It was not as simple as coding for a serial machine, but we were never completely baffled in our search for efficient parallel algorithms.

The support routines were more challenging for certain of the authors. We found that it was not a trivial exercise to generate an efficient particle transport routine. Likewise, the automatic rezone routine presented certain difficulties in handling the rezone in the radial direction. All of these problems were eventually overcome and satisfactory algorithms were developed.

We observed a B6500 in action at the University of Illinois as it compiled GLYPNIR programs into ILLIAC machine language. For our SHELL code the overall time turned out to be a little better than 100 cards per minute of processor time. This is for the complete cycle from GLYPNIR source code through ASK and

the collector and loader. The SHELL code consisted of about 1200 GLYPNIR cards and took approximately 12 minutes of B6500 processor time. The speed of a B6500, we understand, can be greatly increased by enlarging its memory size. We hope that the B6500 or similar computer that drives the ILLIAC IV will be equipped with a large memory.

We anticipate that a running SHELL code will produce too much data to be stored on magnetic tape, but the laser store as advertised should handle the problem quite nicely. We estimate that approximately 8×10^9 bits of data will be sufficient to describe the complete time history of a 500,000-zone problem. To process and analyze this volume of data, analysis routines will be written in GLYPNIR for execution on ILLIAC.

APPENDIX I

COMPUTER LISTING OF THE BASIC STRIPPED VERSION OF SAP IN FORTRAN

```
   B 5 5 0 0    F O R T R A N    C O M P I L A T I O N     X.15,   SATURDAY
                                                        START OF SEGMENT *
      COMMON CS(64),EPG(64),P(64),Q(64),RHO(64),ZM(64),U(64),X(64)
      COMMON DESCT(12)
      COMMONT(64)
C     READ STATEMENTS
      READ 61,(DESCT(I),I=1,12)
C
      JFIN=60
      READ 63, PNUM,CO,C1,WW,TS,CKP,UMIN,QMIN,TIME,BOMAS,HOB,ANGLE,DXMUL
      IF(UMIN.EQ.0.) UMIN= 1.0E-10
      PRINT 64,(DESCT(I),I=1,12)
      PRINT 66,NC,NTEDIT,JFIN,JCYCS,N,ITIME,LTAPE,PNUM,CO,C1,WW,TS
      PRINT 67,CKP,UMIN,QMIN,TIME,BOMAS,HOB,ANGLE
      XMINUS=0.
      JSTAR=JFIN
      I=0
      X(1)=0.0
      U(1)=0.0
      J= 1
      DO 16 K=1,JFIN
C     DX IS THE DELTA X FOR THE ZONES(CM)
C     EZZ IS THE ENERGY FOR THE ZONES(ERGS/GM)
C     RZZ IS THE DENSITY FOR THE ZONES(GM/CM3)
C     UZZ IS THE VELOCITY FOR THE ZONES(CM/SEC)
C     I IS THE MAX ZONE NUMBER FOR THESE CONDITIONS
      IF(K.LE.I) GO TO 11
8     READ 69,DX,EZZ,RZZ,UZZ,I
      IF(UZZ.NE.0) JSTAR= 1
      IF(JSTAR.GT.JFIN) JSTAR= JFIN
11    X(J)=XMINUS+DX
      IF(EZZ) 700,700,701
  700 RHO(J)=1.225E-3
      EPG(J)=2.E9
      GO TO 702
  701 RHO(J)=RZZ
      EPG(J)=EZZ
  702 GMONE    =0.4
      P(J)=GMONE    *RHO(J)*EPG(J)
      CS(J)=SQRT (P(J)*1.4/RHO(J))
      Q(J)=0.
      U(J)=UZZ
      ZM(J)=RHO(J)*(X(J)**3-XMINUS**3)
      XMINUS=X(J)
      IF(MOD(K,50).EQ.1) PRINT 2004, DESCT,PNUM
      PRINT 2005,  K,DX,X(J),EPG(J),P(J),RHO(J),U(J),T(J),ZM(J),K
      J= J+1
16    CONTINUE
C
      CO2=CO*CO
      TC1=2.*C1
      FCO2=4.*CO2
      DTZJM=WW*X(1)/(SQRT(P(1)*1.4/RHO(1))  )/2.
      DTC=DTZJM
23    DELT=DTC
      DTC=DTZJM
30    DTZJM=1.E30
      XJM1=0.
      XMINUS=0.
      UMINUS=0.
```


Preceding page blank

```
      DO 341 J =1,JSTAR
      U(J)=U(J)+DELT*((P(J)+Q(J)-Q(J+1)-P(J+1))/(RHO(J)*(X(J)-XJM1)+RHO(
     1J+1)*(X(J+1)-X(J)))+G)
      XJM1=X(J)
      IF (ABS (U(J))-UMIN) 33,33,31
31    X(J)=X(J)+U(J)*DTC
      IF(J -JSTAR)34,32,34
 32   JSTAR=J +1
      GO TO 341
33    U(J)=0.
34     CONTINUE
 341  CONTINUE
      PMAX=0.
35    JSTP1=JSTAR+1
      DO 471 J =1,JSTP1
      RHON=ZM(J)/(X(J)**3-XMINUS**3)
      DV=1./RHON-1./RHO(J)
      IF (DV) 36,36,38
36    DU=U(J)-UMINUS
      IF (DU) 37,38,38
37    Q(J)=RHON*(DU*CQ2-C1*CS(J))*DU
      IF (Q(J)-QMIN) 38,38,39
38    Q(J)=0.
      CRNT=0.
      GO TO 40
39    CRNT=TC1*CS(J)-FCQ2*DU
 40   GMONE=0.4
      P2=GMONE*RHON*EPG(J)
      E1=AMAX1(EPG(J)-DV*P(J),0.)
      P1=GMONE*RHON*E1
      EPG(J)=AMAX1(EPG(J)-(P2+P(J)+Q(J)+Q(J))*DV/(2.-(P1-P2)/P(J)),0.)
      P(J)=GMONE*RHON*EPG(J)
      FUDGE=.001*RHON
      IF (DV) 41,42,42
41    RHOF=RHON+FUDGE
      GO TO 43
42    RHOF=RHON-FUDGE
43     CONTINUE
      PFUDGE=GMONE*RHOF*EPG(J)
      CS(J)=SQRT ((P(J)-PFUDGE)/(RHON-RHOF))
      CRNT=CRNT+CS(J)
      CRNT=CRNT+1.E-2
      DTZJ=WR*(X(J)-XMINUS)/CRNT
      IF (DTZJM-DTZJ) 45,45,44
44    DTZJM=DTZJ
45    RHO(J)=RHON
      UMINUS=U(J)
      IF (P(J)-PMAX) 47,46,46
46    PMAX=P(J)
47    XMINUS=X(J)
 471  CONTINUE
      TIME=TIME+DTC
C     EXIT CONTROL
      JSTAR=MIN0(JSTAR,JFIN)
      DO 601 J =1,JSTAR
      IF(MOD(J ,50)-1)601,59,601
59    PRINT 84,(DESCT(I),I=1,12)
      PRINT 98,PNUM,N,TIME,DTC
601   PRINT 99,J ,X(J),U(J),Q(J),P(J),EPG(J),CS(J),RHO(J),T(J),J
      N=N+1
      IF(N.LE.10)GO TO 23
```

```
      STOP 777                                                            F
C                                                                    0259 F
C        FORMATS FOR SAP                                             0260 F
61     FORMAT (12A6)                                                 0274 F
63     FORMAT (8E10.0)                                               0276 F
64     FORMAT (1H1,12A6)                                             0277 F
66     FORMAT (/7H0SYMBOL,3X,10HDEFINITION,66X,5HVALUE/3H0NC,7X,36HNUMBER 0280 F
      1 OF DIFFERENT JFLTS AND NDUMPS,40X,I6/7H NTFUIT,3X,30HNUMBER OF DI 0281 F
      2FFERENT EDIT TIMES,40X,I6/5H JFIN,5X,26HNUMBER OF ZONES IN PROBLEM 0282 F
      3,50X,I6/6H JCYCS,4X,32HMAXIMUM NUMBER OF PROBLEM CYCLES,44X,I6/2H  0283 F
      4N,6X,26HCYCLE NUMBER OF INPUT DATA,50X,I6/6H ITIME,4X,28HPOWER OF 0284 F
      5TEN FOR FIRST OUTPUT,46X,I6/6H LTAPE,4X,41HUSED TO DESIGNATE INPUT 0285 F
      6TAPES FOR RESTART,35X,I6/5H PNUM,5X,14HPROBLEM NUMBER,60X,F6.3/3H  0286 F
      7CO,7X,24HQUADRATIC VISCOSITY TERM,46X,1PE12.3/3H C1,7X,21HLINEAR V 0287 F
      8ISCOSITY TERM,49X,E12.3/3H WW,7X,35HMULTIPLIES THE CALCULATED TIME 0288 F
      9 STEP,35X,E12.3/3H TS,7X,20HTIME OF PROBLEM STOP,50X,E12.3)        0289 F
67     FORMAT (4H CKP,6X,41HDISTANCE OF MAX. PRESSURE AT PROBLEM STOP,29X 0290 F
      1,1PE12.3/5H UMIN,5X,27HMINIMUM VELOCITY IN PROBLEM,43X,E12.3/5H QM 0291 F
      2IN,5X,20HMINIMUM Q IN PROBLEM,50X,E12.3/                          0292 F
      3                         5H TIME,5X,24HTIME OF START OF PROBLEM,46X,E1 0293 F
      42.3/                                                        6H    0294 F
      6 BOMAS,4X,12HMASS OF CORE,58X,E12.3/                              0295 F
      74H HOR,6X,27HHEIGHT OF ZERO POINT IN CM.,43X,E12.3/6H ANGLE,4X,36H 0296 F
      8DIRECTION OF RUN (DEGREES,POS. UPWARD),32X,E12.3)                 0297 F
                                                                  SEGMENT
69     FORMAT (4E12.4,I5)                                            0299 F
98     FORMAT (10H0PROB. NO.F8.3,3X,2HN=I4,3X,5HTIME=E14.5,3X,4HDTC=E14.5 0316 F
      1/4H0  J,7X,1HX,12X,1HU,12X,1HQ,12X,1HP,12X,1HE,10X,4H CS ,10X,3HRH 0317 F
      2O,9X,6HP/PO-1,6X,1HJ/)                                        0318 F
99     FORMAT (1H I4,1P8E13.5,I4)                                    0319 F
2004   FORMAT(1H1,12A6/1H /5X,7HPNUM = ,F10.2/1H /3X,1HJ,12X,2HDX,13X, 0270 F
      1  1HX,11X,3HEPG,13X,1HP,11X,3HRHO,13X,1HU,13X,1HT,12X,2HT/M,3X,1HJ, 0271 F
      2  /1H )                                                       0272 F
2005   FORMAT(1H I3,8E14.5,I4)                                       0273 F
       END                                                           0320 F
                                                                  SEGMENT
```

APPENDIX II

COMPUTER LISTING OF *FORTRAN SAP

```
      COMMON/MAIN/Z(64),U(64),P(64),Q(64),RHO(64),X(64),ZM(64),
     1  EPG(64),CS(64),T(64),RDRT(64)
      COMMON/TEMP/RHON(64),DV(64),CRNT(64),GMONE(64),P2(64),E1(64),
     1P1(64),RHOF(64),DTZJ(64)
      EQUIVALENCE (PROB,Z(1)),((N,Z(2)),(TIME,Z(3)),(DFLT,Z(4)),
     1(DTC,Z(5)),(JSTAR,Z(6)),(JSTP1,Z(7)),(MJSTAR,Z(8)),(MJSTP1,Z(9)),
     2(JFIN,Z(10)),(PMAX,Z(11)),(JPMAX,Z(12)),(XPMAX,Z(13)),(CO2,Z(14)),
     3(C1,Z(15)),(TC1,Z(16)),(FCO2,Z(17)),(DTZJM,Z(18))
      FILE INPUT(0,704) SERIAL
      FILE OUTPUT(0,704) SERIAL
      BINARY MJSTAR(64), MJSTP1(64)
 23   DELT=DTC
      DTC=DTZJM
      MJSTAR.(0,63)=0
      MJSTAR.(1,JSTAR)=1
      DO 341 MODE=MJSTAR
  341 U(*)=U(*)+DELT*((P(*)+Q(*)-Q(*+1)-P(*+1)))/ (RHO(*)*(X(*)-X(*-1))+
     1RHO(*+1)*(X(*+1)-X(*)))
      DO 351 MODE=MJSTAR
      IF(ABS(U(*)).LT.UMIN) U(*)=0.
  351 X(*)=X(*)+DTC*U(*)
      IF(U(JSTAR).GT.0.) JSTAR=JSTAR+1
      IF(JSTAR.GE.JFIN)  STOP 351
      JSTP1=JSTAR+1
      MJSTP1.(0,63)=0
      MJSTP1.(1,JSTP1)=1
      DO 471 MODE=MJSTP1
      RHON(*)=ZM(*)/(X(*)**3-X(*-1)**3)
      DV(*)=1./RHON(*)-1./RHO(*)
      DU(*)=U(*)-U(*-1)
      Q(*)=0.
      IF(DV(*).LE.0.) Q(*)=RHON(*)*(AMIN1(DU(*),0.)*CO2-C1*CS(*))*AMIN1(
     1DU(*),0.)
      IF(Q(*).LE.QMIN)Q(*)=0.
      CRNT(*)=0.
      IF(Q(*).NE.0.) CRNT(*)=TC1*CS(*)-FCO2*DU(*)
      GMONE(*)=0.4
      P2(*)=GMONE(*)*RHON(*)*EPG(*)
      E1(*)=AMAX1((EPG(*)-DV(*)*P(*)),0.)
      P1(*)=GMONE(*)*RHON(*)*E1(*)
      EPG(*)=AMAX1((EPG(*)-(P2(*)+P(*)+Q(*)+Q(*))*DV(*)/(2.+(P2(*)-P1(*)
     1)/P(*))),0.)
      P(*)=GMONE(*)*RHON(*)*EPG(*)
      P2(*)=+0.001
      IF(DV(*).GE.0.) P2(*)=-P2(*)
      RHOF(*)=RHON(*)*(1.+P2(*))
      PFUDGE(*)=GMONE(*)*RHOF(*)*EPG(*)
      CS(*)=SQRT((P(*)-PFUDGE(*))/(RHON(*)-RHOF(*)))
      DTZJ(*)=WW*(X(*)-X(*-1))/ (CRNT(*)+1.E-2+CS(*))
      IF(DTZJ(J).LT.DTZJM ) DTZJM=DTZJ(J)
      RHO(*)=RHON(*)
      PMAX=AMAX1(P(J),PMAX)
  471 CONTINUE
      TIME=TIME+DTC
C     EXIT CONTROL
      WRITE (OUTPUT) Z
      GO TO 23
      STOP 7777
      END
```

APPENDIX III

COMPUTER LISTING GLYPNIR SAP62

```
$CODE LIST ZIP DRUGA SUMRY
$NCDCNT PRODUCTION SAVE
BEGIN
PREAL SUBROUTINE SORT AS RGA(PREAL Y AS RGA);
BEGIN
$INCINS 110 ASKRATCH 14
$$TAPE2= GLYPNIR/GSORT SERIAL;
END;
  SUBROUTINE PUTCPR(CREAL X,PREAL OUT Z,CINT I);
 BEGIN
  FOR ALL PEN=I DO Z+X;
   END;
  SUBROUTINE PUTCPI(CINT X,PREAL OUT Z,CINT I);
 BEGIN
 CU REAL A;
 A+X;
 FOR ALL PEN=I DO Z+A;
   END;
 CREAL PROB,TIME,DELT,DTC,PMAX,XPMAX,CO2,C1,TC1,FCO2,DTZJM,QMIN,UMIN,WW;
   CINT     N,JSTAR,JSTP1,JFIN,ZPROB,ZN,ZTIME,ZDELT,ZDTC,ZJSTAR,ZJSTP1;
    CINT    ZJFIN,ZPMAX,ZJPMAX,ZXPMAX,ZCO2,ZC1,ZTC1,ZFCO2,ZDTZJM,ZQMIN;
CINT JPMAX,ZUMIN,ZWW,ZNSTOP,NSTOP;
  PREAL Z,U,P,Q,RHO,X,ZM,EPG,CS,T,RDRT,X3,RHON,DV,CRNT,GMONE,P2,F1,P1;
PREAL PDU,RHOF,DTZJ,DU,PFUDGE;
 LABEL START,AGAIN,FIN,STOP;
% VARIABLES HAVE BEEN DECLARED
%SET UP INPUT AND OUTPUT FILES
FILE I4DISK2="SAP"/"BTOI"(11 ROWS FULL);
FILE I4DISK1="SAP"/"ITOB"(11 ROWS FULL);
OPEN(I4DISK1,0,1);
OPEN(I4DISK2,0,1);
 START:
SIMREAD(I4DISK2,Z);
SIMWRITE(I4DISK1,Z);
  ZPROB+0;
  ZN+1;
  ZTIME+2;
  ZDELT+3;
  ZDTC+4;
  ZJSTAR+5;
 ZJSTP1+6;
  ZQMIN+7;
  ZUMIN+8;
  ZJFIN+9;
  ZPMAX+10;
  ZJPMAX+11;
  ZXPMAX+12;
  ZCO2+13;
  ZC1+14;
  ZTC1+15;
  ZFCO2+16;
  ZDTZJM+17;
  ZWW+18;
  ZNSTOP+19;
 DTZJ+100000.0;
      QMIN+GRABONE(Z,ZQMIN);
      WW+GRABONE(Z,ZWW);
      DTZJM+GRABONE(Z,ZDTZJM);
      JSTAR+GRABONE(Z,ZJSTAR);
      PROB+GRABONE(Z,ZPROB);
```

```
      N←GRABONE(Z,ZN);
      TIME←GRABONE(Z,ZTIME);
      DTC←GRABONE(Z,ZDTC);
      UMIN←GRABONE(Z,ZUMIN);
      JFIN←GRABONE(Z,ZJFIN);
      CO2←GRABONE(Z,ZCO2);
      C1←GRABONE(Z,ZC1);
      TC1←GRABONE(Z,ZTC1);
      NSTOP←GRABONE(Z,ZNSTOP);
      FCO2←GRABONE(Z,ZFCO2);
 AGAIN: DELT←DTC ;
      DTC←DTZJM;
      MODE←(PEN≤JSTAR)AND(NOT BOOLEAN(-0));
      U←U+DELT×((P+Q)-RTL(1,TRUE,(P+Q)))/(RHO×(X-RTL(-1,TRUE,X))+RTL(1
      ,TRUE,RHO)×(RTL(1,TRUE,X)-X));
      IF(ABS(U)<UMIN) THEN U←0.;
      X←X+DTC×U;
      IF(GRABONE(U,JSTAR)>0.) THEN JSTAR←JSTAR+1;
      IF(JSTAR>JFIN) THEN GO TO STOP;
      JSTP1←JSTAR+1;
      MODE←SHIFTR(1,MODE OR BOOLEAN(-0));
      X3←X×X×X;
      RHON←ZM/(X3-RTL(-1,TRUE,X3));
      DV←(RHO-RHON)/(RHON×RHO);
      DU←(U-RTL(-1,TRUE,U));
      Q←0.;
      PDU←DU;
      IF(DU>0.0) THEN PDU←0.0;
      IF(DV<0.) THEN Q←RHON×(PDU×CO2-C1×CS)×PDU;
      IF(Q≤QMIN) THEN Q←0.;
      CRNT←0.0;
      IF(Q>0.0) THEN CRNT←TC1×CS-FCO2×PDU;
      GMONE←0.4;
      P2←GMONE×RHON×EPG;
      E1←(EPG-DV×P);
      IF(E1<0.) THEN E1←0.;
      P1←GMONE×RHON×E1;
      EPG←EPG-(P2+P+Q+Q)×DV/(2.+(P2-P1)/P);
      IF(EPG<0.) THEN EPG←0.;
      P←GMONE×RHON×EPG;
      RHOF←RHON×1.001;
      IF(DV>0.0) THEN RHOF←RHON×.999;
      PFUDGE←GMONE×RHOF×EPG;
      CS←SQRT((P-PFUDGE)/(RHON-RHOF));
      DTZJ←WW×(X-RTL(-1,TRUE,X))/(CRNT+.01+CS);
      DTZJM←MIN(DTZJ);
      RHO←RHON;
      PMAX←MAX(P);
      TIME←TIME+DTC;
MODE←TRUE;
      PUTCPR(PROB,Z,ZPROB);
      PUTCPR(DTZJM,Z,ZDTZJM);
      PUTCPR(PMAX,Z,ZPMAX);
      PUTCPR(TIME,Z,ZTIME);
      PUTCPR(DTC,Z,ZDTC);
      PUTCPI(N,Z,ZN);
      PUTCPI(JSTAR,Z,ZJSTAR);
      PUTCPR(DELT,Z,ZDELT);
      PUTCPR(XPMAX,Z,ZXPMAX);
      PUTCPI(JPMAX,Z,ZJPMAX);
      PUTCPI(JSTP1,Z,ZJSTP1);
      PUTCPI(NSTOP,Z,ZNSTOP);
SIMWRITE(I4DISK1,Z);
      N←N+1;
      IF(N<NSTOP) THEN GO TO AGAIN;
FIN:STOP:END.
```

APPENDIX IV

COMPUTER LISTINGS OF PROGRAMS I, SAP62 INPUT, AND II, SAP62 OUTPUT

```
$CARD LIST SINGLE
FILE  1=SAP/PTOT,UNIT=DISK,SAVE=1,LOCK,SERIAL,AREA=4,
     1BLOCKING=1,RECORD=540,BUFFER=2
      COMMON/MATH/Z(64),U(64),P(64),Q(64),RHO(64),X(64),ZM(64),
     1  EPG(64),CS(64),T(64),RDRT(64)
      REAL DESCT(12)
      EQUIVALENCE (PNUM,Z(1)), (N,Z(2)),(TIME,Z(3)),(DELT,Z(4)),
     1(DTC,Z(5)),(JSTAR,Z(6)),(JSTP1,Z(7)),(QMIN  ,Z(8)),(UMIN  ,Z(9)),
     2(JFIN,Z(10)),(PMAX,Z(11)),(JPMAX,Z(12)),(XPMAX,Z(13)),(C02,Z(14)),
     3(C1,Z(15)),(TC1,Z(16)),(FC02,Z(17)),(DTZJM,Z(18)),( WW,Z(19)),
     4(NSTOP,Z(20)),(DESCT(1),Z(52))
C     READ STATEMENTS                                                   0019
      READ 81,(DESCT(I),I=1,12)                                         0020
C                                                                       0016
      JFIN=60
      READ 83, PNUM,CO,C1,WW,TS,CKP,UMIN,QMIN,TIME,BOMAS,HOB,ANGLE,DXMUL 0022
      IF(UMIN.EQ.0.) UMIN= 1.0E-10
      PRINT 84,(DESCT(I),I=1,12)                                        0029
      PRINT 86,NC,NTEDIT,JFIN,JCYCS,N,ITIME,LTAPE,PNUM,CO,C1,WW,TS      0031
      PRINT 87,CKP,UMIN,QMIN,TIME,BOMAS,HOB,ANGLE                       0032
      XMINUS=0.                                                         0041
      U(1)=0.0
      X(1)=0.0
      JSTAR=JFIN
      I= 0
      J= 2
      DO 16 K=1,JFIN
C     DX IS THE DELTA X FOR THE ZONES(CM)
C     EZZ IS THE ENERGY FOR THE ZONES(ERGS/GM)
C     RZZ IS THE DENSITY FOR THE ZONES(GM/CM3)
C     UZZ IS THE VELOCITY FOR THE ZONES(CM/SEC)
C     I IS THE MAX ZONE NUMBER FOR THESE CONDITIONS
      IF(K.LE.I) GO TO 11
      READ 89,DX,EZZ,RZZ,UZZ,I
      IF(UZZ.NE.0) JSTAR= I
      IF(JSTAR.GT.JFIN) JSTAR= JFIN
11    X(J)=XMINUS+DX
      IF(EZZ.GT.0) GO TO 701
      RHO(J)=1.225E-3
      EPG(J)=2.E9
      GO TO 702
  701 RHO(J)=RZZ
      EPG(J)=EZZ
  702 GMONE   =0.4
      P(J)=GMONE   *RHO(J)*EPG(J)
      CS(J)=SQRT (P(J)*1.4/RHO(J))
      Q(J)=0.
      U(J)=UZZ
      ZM(J)=RHO(J)*(X(J)**3-XMINUS**3)
      XMINUS=X(J)
      IF(MOD(K,50).EQ.1) PRINT 2004, DESCT,PNUM
      PRINT 2005,  K,DX,X(J),EPG(J),P(J),RHO(J),U(J),T(J),ZM(J),K
      J= J+1
C
16    CONTINUE
      C02=CO*CO
      TC1=2.*C1
      FC02=4.*C02
      DTZJM=WW*X(2)/(SQRT(P(2)*1.4/RHO(2))  ) /2.
      DTC=DTZJM
```

```
      Z(1)=PNUM
      Z(6)=JSTAR
      Z(10)=JFIN
      Z(14)=CO2
      Z(15)=C1
      Z(16)=TC1
      Z(17)=FCO2
      Z(18)=DTZ.IM
      Z(19)=WW
      NSTOP=10
      CALL WRTIAD(Z,704)
      STOP
81    FORMAT (12A6)                                                     0274
83    FORMAT (8F10.0)                                                   0276
84    FORMAT (1H1,12A6)                                                 0277
86    FORMAT (/7HOSYMBOL,3X,10HDEFINITION,66X,5HVALUE/3HONC,7X,36HNUMBER 0280
     1 OF DIFFERENT JPLTS AND NDUMPS,40X,I6/7H NTEDIT,3X,30HNUMBER OF DI0281
     2FFERENT EDIT TIMES,46X,I6/5H JFIN,5X,26HNUMBER OF ZONES IN PROBLEM0282
     3,50X,I6/6H JCYCS,4X,32HMAXIMUN NUMBER OF PROBLEM CYCLES,44X,I6/2H 0283
     4N,8X,26HCYCLE NUMBER OF INPUT DATA,50X,I6/6H ITIME,4X,28HPOWER OF 0284
     5TEN FOR FIRST AUIDT,48X,I6/6H LTAPE,4X,41HUSED TO DESIGNATE INPUT 0285
     6TAPES FOR RESTART,35X,I6/5H PNUM,5X,14HPROBLEM NUMBER,60X,F8.3/3H 0286
     7CO,7X,24HQUADRATIC VISCOSITY TERM,46X,1PE12.3/3H C1,7X,21HLINEAR V0287
     8ISCOSITY TERM,49X,F12.3/3H WW,7X,35HMULTIPLIES THE CALCULATED TIME0288
     9 STEP,35X,F12.3/3H TS,7X,20HTIME OF PROBLEM STOP,50X,F12.3)       0289
87    FORMAT (4H CKP,6X,41HDISTANCE OF MAX. PRESSURE AT PROBLEM STOP,29X0290
     1,1PE12.3/5H UMIN,5X,27HMINIMUN VELOCITY IN PROBLEM,43X,F12.3/5H QM0291
     2IN,5X,20HMINIMUN Q IN PROBLEM,50X,F12.3/                          0292
     3                          5H TIME,5X,24HTIME OF START OF PROBLEM,46X,E10293
     42.3/                                                            6H0294
     6 BOMAS,4X,12HMASS OF CORE,58X,F12.3/                              0295
     74H HOB,6X,27HHEIGHT OF ZERO POINT IN CM.,43X,F12.3/6H ANGLE,4X,38H0296
     8DIRECTION OF RUN (DEGREES,POS. UPWARD),32X,F12.3)                 0297
89    FORMAT (4F12.4,I5)                                                0299
2004  FORMAT(1H1,12A6/1H /5X,7HPNUM = ,F10.2/1H /3X,1HJ,12X,2HDX,13X,   0270
     1  1HX,11X,3HFPG,13X,1HP,11X,3HRHO,13X,1HU,13X,1HT,12X,2HZM,3X,1HJ, 0271
     2  /1H )                                                           0272
2005  FORMAT(1H ,I3,8E14.5,I4)                                          0273
      END
      SUBROUTINE WRTIAD(Z,N)
      REAL A(540),Z(N)
      PRINT 99
99    FORMAT(1H1)
      LL= 0
      NN= N
5     K= NN
      IF(K.GT.256) K=256
      DO 11 I=1,K
      L= LL+I
      IF(L.GE.52.AND.L.LE.63) GO TO 50
      CALL R5IAF(Z(L),A(2*I-1),A(2*I))
      GO TO 10
50    CALL  RIAT(Z(L),A(2*I-1),A(2*I))
10    IF(Z(I).EQ.0.) GO TO 11
      PRINT 100, L,Z(L),Z(L),A(2*I-1),A(2*I)
11    CONTINUE
100   FORMAT(I10,F30.6,E20.6,2O20)
      WRITE(1) (A(I),I=1,540)
      NN= NN-K
      IF(NN.EQ.0) RETURN
      LL= LL+256
```

```
      GO TO 5
      END
      SUBROUTINE B5I4F(A,B,C)
      EQUIVALENCE (XX,II)
C
      XE=CONCAT(0,A,41,2,7)
      IF(XE.EQ.0) GO TO 10
      CALL BI4F(A,B,C)
      RETURN
C
10    XX= A
      X= II
      X= X+0.1
      X= X-0.1
      CALL BI4F(X,B,C)
      RETURN
      END
      SUBROUTINE BI4F(A,B,C)
      EQUIVALENCE (XE,IE8)
      DATA IBIAS/040000/
      IF(A.NE.0) GO TO 10
      B= 0
      C= 0
      RETURN
C
10    XS=CONCAT(0,A,47,2,1)
      XE=CONCAT(0,A,42,3,6)
      IF(XS.NE.0)   IE8=-IE8
      IE8=-IE8
      B9=CONCAT(0,A,47,9,1)
      B10=CONCAT(0,A,47,10,1)
      IE8E=2
      IF(B10.NE.0)   IE8E=1
      IF(B9.NE.0)   IE8E=0
      IE=3*IE8+IE8E
      IE= IBIAS-IE+39
      BB=CONCAT(0,A,16,1,1)
      BB=CONCAT(BB,IE,17,33,15)
      B=CONCAT(BB,A,32,9+IE8E,16)
      C=CONCAT(0,A,16,25+IE8E,23-IE8E)
      RETURN
      END
      SUBROUTINE BI4I(A,B,C)
      BB=CONCAT( 0,A,16,1,1)
      B=CONCAT(BB,A,41,9,7)
      C=CONCAT(0,A,16,16,32)
      RETURN
      END
   SAP STRIPPED AND READY FOR ACTION
    1.            1.8        0.5        0.2        30.     1.E30        0.        0.0
     .0000          0.         0.        0.        0.
          1.         2.E13       1.E-3        6.E05   30
    1.         0.         0.        0.             63
```

```
$CARD LIST SINGLE
FILE   2=SAP/ITOR,UNIT=DISK,SERIAL,BLOCKING=1,RECORD=540
       COMMON/MAIN/Z(64),U(64),P(64),Q(64),RHO(64),X(64),ZM(64),
      1  FPG(64),CS(64),T(64),RDPT(64)
       REAL DESC T(12)
       EQUIVALENCE (PNUM,Z(1)), (N,Z(2)),(TIME,Z(3)),(DELT,Z(4)),
      1(DTC,Z(5)),(JSTAR,Z(6)),(JSTP1,Z(7)),(QMIN  ,Z(8)),(UMIN  ,Z(9)),
      2(JFIN,Z(10)),(PMAX,Z(11)),(JPMAX,Z(12)),(XPMAX,Z(13)),(CO2,Z(14)),
      3(C1,Z(15)),(TC1,Z(16)),(FCO2,Z(17)),(DTZJM,Z(18)),( WN,Z(19)),
C
      4(DESCT(1),Z(52))
 100   CALL REDIAD(Z,704)
       PRINT 999
999    FORMAT(1H1)
       DO 199 I=1,704
       IF(Z(I).EQ.0.) GO TO 199
198    PRINT 200, I,Z(I),Z(I),Z(I)
199    CONTINUE
200    FORMAT(I10,F20.6,E30.6,O40)
       N=Z(2)
       JSTAR=Z(6)
       DO 601 JJ=1,JSTAR
       J= JJ+1
       IF(MOD(JJ,50).NE.1) GO TO 601
       PRINT 84,(DESCT(I),I=1,12)
       PRINT 98,PNUM,N,TIME,DTC
601    PRINT 99,JJ,X(J),U(J),Q(J),P(J),FPG(J),CS(J),RHO(J),   T(J),JJ
       GO TO 100
C
C         FORMATS FOR SAP
84     FORMAT (1H1,12A6)
98     FORMAT (10HOPROB. NO.F8.3,3X,2HN=I4,3X,5HTIME=F14.5,3X,4HDTC=F14.5
      1/4HO   J,7X,1HX,12X,1HU,12X,1HQ,12X,1HP,12X,1HF,10X,4H CS ,10X,3HRH
      2O,9X,6HP/PO-1,6X,1HJ/)
99     FORMAT (1H I4,1P8E13.5,I4)
       END
       SUBROUTINE REDIAD(Z,N)
       REAL ZZZ(1620),Z(N)
       LOGICAL FIRST
       DATA FIRST /.TRUE./
C
       IF(FIRST) REWIND 2
       FIRST= .FALSE.
       READ (2,END=200) (ZZZ(I),I=1,540)
       READ(2) (ZZZ(I+540),I=1,540)
       READ(2) (ZZZ(I+1080),I=1,540)
       DO 10 I=1,256
       IF(I.GE.52.AND.I.LE.63) GO TO 50
       CALL TABF(ZZZ(2*I-1),ZZZ(2*I),Z(I))
       GO TO 10
50     CALL TABT(ZZZ(2*I-1),ZZZ(2*I),Z(I))
10     CONTINUE
       DO 11 I=1,256
11     CALL TABF(ZZZ(2*I+539),ZZZ(2*I+540),Z(I+256))
       DO 12 I=1,192
12     CALL TABF(ZZZ(2*I+1079),ZZZ(2*I+1080),Z(I+512))
       RETURN
C
200    PRINT 201
201    FORMAT(24H1END OF FILE ON I4 DISK.)
```

```
      STOP
      END
      SUBROUTINE I4BF(A,B,C)
      INTEGER O100
      DATA IBIAS,O100/O40000,O100/
      EQUIVALENCE(XE,IE)
C
C
      XE= CONCAT(O,A,33,17,15)
      IE= IBIAS+39-IE
C        IE8 IS POWER OF 8
10    IE8= IE/3
C        IE8E IS RT SHIFT OF MANTISSA AT IE8
      IE8E= IE-3*IE8
      IF(IE8E.GE.0) GO TO 20
      IE8E= IE8E+3
      IE8= IE8-1
20    IE8= -IE8
      IF(IE8.LT.0) IE8= O100-IE8
C        PICK UP SIGN
      OUT= CONCAT(O,A,1,16,1)
C        PICK UP EXPONENT
      OUT= CONCAT(OUT,IE8,2,41,7)
C        PICK UP TOP OF MANTISSA
      OUT= CONCAT(OUT,A,9+IE8E,32,16)
C        PICK UP BOTTOM OF MANTISSA
      C  = CONCAT(OUT,B,25+IE8E,16,23-IE8E)
      RETURN
      END
      SUBROUTINE I4BI(A,B,C)
      CC=CONCAT(O,A,1,16,1)
      CC=CONCAT(CC,A,9,41,7)
      C=CONCAT(CC,B,16,16,32)
      RETURN
      END
```

APPENDIX V

COMPUTER LISTING OF THE SAP62 CHANGEZ CODE

```
$CARD LIST
FILE  1=SAP/ITOP,UNIT=DISK,SERIAL,BLOCKING=1,RECORD=540
FILE  2=SAP/BTOT,UNIT=DISK,SERIAL,BLOCKING=1,RECORD=540
      COMMON Z(704)
      COMMON /FLAGS/ IIT,IOT,CY,DMY(1620)
C
C
      IIT= 1
      IOT= 2
      CY= 3
C
C
      CALL REDIAD(Z,704)
20    READ 100,I,F
100   FORMAT(I5,F15.0)
      IF(I.LE.0) GO TO 40
      Z(I)= F
      GO TO 20
C
40    CALL WRTIAD(Z,704)
      STOP
      END
      SUBROUTINE REDIAD(Z,N)
      REAL Z(N)
      COMMON /FLAGS/ IIT,IOT,CY,ZZZ(1620)
C
      REWIND IIT
101   READ (IIT,END=200) (ZZZ(I),I=1,540)
      READ(IIT) (ZZZ(I+540),I=1,540)
      READ(IIT) (ZZZ(I+1080),I=1,540)
      DO 10 I=1,256
      IF(I.GE.52.AND.I.LE.63) GO TO 50
      CALL I4BF(ZZZ(2*I-1),ZZZ(2*I),Z(I))
      GO TO 10
50    CALL I4BI(ZZZ(2*I-1),ZZZ(2*I),Z(I))
10    CONTINUE
      IF(Z(2) -CY) 101,102,103
102   DO 11 I=1,256
11    CALL I4BF(ZZZ(2*I+539),ZZZ(2*I+540),Z(I+256))
      DO 12 I=1,192
12    CALL I4BF(ZZZ(2*I+1079),ZZZ(2*I+1080),Z(I+512))
      RETURN
C
103   PRINT 104,Z(2),CY
104   FORMAT(18H1HAVE FOUND CYCLE F5.0,19H LOOKING FOR CYCLE F5.0)
      STOP
200   PRINT 201
201   FORMAT(24H1END OF FILE ON I4 DISK.)
      STOP
      END
      SUBROUTINE I4BF(A,B,C)
      INTEGER B100
      DATA IBIAS,B100/040000,0100/
      EQUIVALENCE(XE,IE)
C
C
      XE= CONCAT(0,A,33,17,15)
      IE= IBIAS+39-IE
C        IE8 IS POWER OF 8
10    IE8= IE/3
```

```
C          IEBE IS RT SHIFT OF MANTISSA AT IE8
      IE8E= IE-3*IE8
      IF(IE8E.GE.0) GO TO 20
      IE8E= IE8E+3
      IE8= IE8-1
20    IE8= -IE8
      IF(IE8.LT.0) IE8= 0100-IE8
C          PICK UP SIGN
      OUT= CONCAT(0,A,1,16,1)
C          PICK UP EXPONENT
      OUT= CONCAT(OUT,IE8,2,41,7)
C          PICK UP TOP OF MANTISSA
      OUT= CONCAT(OUT,A,9+IE8E,32,16)
C          PICK UP BOTTOM OF MANTISSA
      C  = CONCAT(OUT,B,25+IE8E,16,23-IE8E)
      RETURN
      END
      SUBROUTINE I4BI(A,B,C)
      CC=CONCAT(0,A,1,16,1)
      CC=CONCAT(CC,A,9,41,7)
      C=CONCAT(CC,B,16,16,32)
      RETURN
      END
      SUBROUTINE WRTI4D(Z,N)
      REAL Z(N)
      COMMON /FLAGS/ IIT,IOT,CY,A(540),DMY(1080)
      REWIND IOT
      PRINT 99
99    FORMAT(1H1)
      LL= 0
      NN= N
5     K= NN
      IF(K.GT.256) K=256
      DO 10 I=1,K
      L= LL+I
      IF(L.GE.52.AND.L.LE.63) GO TO 50
      CALL B5I4F(Z(L),A(2*I-1),A(2*I))
      GO TO 10
50    CALL  BI4I(Z(L),A(2*I-1),A(2*I))
10    PRINT 100, L,Z(L),Z(L),A(2*I-1),A(2*I)
100   FORMAT(I10,F30.6,E20.6,2020)
      WRITE(IOT) (A(I),I=1,540)
      NN= NN-K
      IF(NN.EQ.0) GO TO 1000
      LL= LL+256
      GO TO 5
C
1000  REWIND IOT
      RETURN
      END
      SUBROUTINE B5I4F(A,B,C)
      EQUIVALENCE (XX,II)
C
      XE=CONCAT(0,A,41,2,7)
      IF(XE.EQ.0) GO TO 10
      CALL BI4F(A,B,C)
      RETURN
C
10    XX= A
      X= II
      X= X+0.1
```

```
      X= X-0.1
      CALL BI4F(X,B,C)
      RETURN
      END
      SUBROUTINE BI4F(A,B,C)
      EQUIVALENCE (XE,IE8)
      DATA IBIAS/040000/
      IF(A.NE.0) GO TO 10
      B= 0
      C= 0
      RETURN
C
10    XS=CONCAT(0,A,47,2,1)
      XE=CONCAT(0,A,42,3,6)
      IF(XS.NE.0)   IE8=-IE8
      IE8=-IE8
      B9=CONCAT(0,A,47,9,1)
      B10=CONCAT(0,A,47,10,1)
      IE8F=2
      IF(B10.NE.0)   IE8F=1
      IF(B9.NE.0)   IE8F=0
      IE=3+IE8+IE8F
      IE= IBIAS-IE+39
      BB=CONCAT(0,A,16,1,1)
      BB=CONCAT(BB,IE,17,33,15)
      B=CONCAT(BB,A,32,9+IE8F,16)
      C=CONCAT(0,A,16,25+IE8F,23-IE8F)
      RETURN
      END
      SUBROUTINE BI4I(A,B,C)
      BB=CONCAT( 0,A,16,1,1)
      B=CONCAT(BB,A,41,9,7)
      C=CONCAT(0,A,16,16,32)
      RETURN
      END

  20     30.
 -1
```

APPENDIX VI

COMPUTER LISTING OF THE SAPN/SKEWED CODE

```
$CODE LIST ZIP DBUGA SUMRY
$       PRODUCTION SAVE
BEGIN
 CREAL PROB,TIME,DELT,DTC,PMAX,XPMAX,CO2,C1,TC1,FCO2,DTZJM,OMIN,UMIN,WW;
 CREAL DTZJMR,PMAXR;
   CINT    N,JSTAR,JSTP1,JFIN,ZPROB,ZN,ZTIME,ZDELT,ZDTC,ZJSTAR,ZJSTP1;
    CINT   ZJFIN,ZPMAX,ZJPMAX,ZXPMAX,ZCO2,ZC1,ZTC1,ZFCO2,ZDTZJM,ZOMIN;
   CINT JPMAX,ZUMIN,ZWW,ZNSTOP,NSTOP,JSX,J,JJ,JMAX;
   PREAL X3,RHON,DV,CRNT,P2,F1,P1,GMONE,POU,RHOF,DTZJ,DU,PFUDGE,Z;
   PE REAL VECTOR U[4],P[4],Q[4],RHO[4],X[4],ZM[4],FPG[4],
        CS[4],T[4],RORT[4];
 LABEL START,AGAIN,    STOP;
 % FOLLOWING CARD FOR SUBROUTINE ADJUSTSLEWED.
  PINT FROMROW,TOROW;BOOLEAN TABLESET;
%
% VARIABLES HAVE BEEN DECLARED
%
%SET UP INPUT AND OUTPUT FILES
%
FILE I4DISK1="SAP"/"ITOB"( 51 ROWS FULL);
FILE I4DISK2="SAP"/"BTOI"( 51 ROWS FULL);
%
PREAL SUBROUTINE SQRT AS RGA(PREAL X AS RGA);
BEGIN
$INCINS 110 ASKRATCH 14
$STAPE2= GLYPNIR/GSQRT SERIAL;
END;
  SUBROUTINE PUTCPR(CREAL X,PREAL OUT Z,CINT I);
 BEGIN
  FOR ALL PEN=I DO Z←X;
   END;
  SUBROUTINE PUTCPI(CINT X,PREAL OUT Z,CINT I);
 BEGIN
 CU REAL A;
 A←X;
 FOR ALL PEN=I DO Z←A;
   END;
 SUBROUTINE SLEW(PCPOINT X, CINT JMAX);
 BEGIN
 CINT J;
 LOOP J←1,1,JMAX DO X[J]←RTL(2×J,MODE,X[J]);
 END;
 SUBROUTINE UNSLEW(PCPOINT X, CINT JMAX);
 BEGIN
 CINT J;
 LOOP J←1,1,JMAX DO X[J]←RTR(2×J,MODE,X[J]);
 END;
 SUBROUTINE ADJUSTSLEWED(PCPOINT X, CINT JMAX);
 BEGIN
 %
 %       DUE TO THE LACK OF OWN VARIABLES AND DATA STATEMENTS IN GLYPNIR
 %       THE FOLLOWING TWO CARDS MUST APPEAR IN THE OUTERMOST BLOCK
 %       IN WHICH ADJUSTSLEWED IS USED.
 %
% PINT FROMROW,TOROW;BOOLEAN TABLESET;
 %TABLESET←FALSE;
 %
 % MODE MUST BE TRUE ON ENTRY TO ADJUSTSLEWED TO PASS ALL ARGS,
 % IT IS TRUE ON EXIT.
 %
 CINT J;LABEL MOVE;
```

```
 IF(TABLESET) THEN GO TO MOVE;
     TABLESET←TRUE;
     TOROW←(64-PEN) DIV 2;
     FROMROW←RTL(1,MODE,TOROW);
     TOROW←RTR(1,MODE,TOROW);
     MODE←BOOLEAN(5555555555555555(16));FROMROW←TOROW;MODE←TRUE;
     TOROW←RTL(1,MODE,TOROW);
 MOVE: LOOP J←0,32,(JMAX-1) DIV 32 DO
     BEGIN
     LABEL MOVEM;
     IF(JMAX≥J+32) THEN GO TO MOVEM;
     MODE←SHIFTR(2×(J+32-JMAX),MODE);
     MOVEM:  X[TOROW+J]←X[FROMROW+J];
     END;
 MODE←TRUE;
 END;%ADJUSTSLEWED
%
%        START OF MAIN PROGRAM.
%
%
 START:
 % FOLLOWING CARD FOR SUBROUTINE ADJUSTSLEWED.
 TABLESET← FALSE;
OPEN(I4DISK1,0,1);
OPEN(I4DISK2,0,1);
      JMAX←A;
SIMREAD(I4DISK2,Z);
SIMWRITE(I4DISK1,Z);
  ZPROB←0;
  ZN←1;
  ZTIME←2;
  ZDFLT←3;
  ZDTC←4;
  ZJSTAR←5;
 ZJSTP1←6;
  ZOMIN←7;
  ZUMIN←8;
  ZJFIN←9;
  ZPMAX←10;
  ZJPMAX←11;
  ZXPMAX←12;
  ZCO2←13;
  ZC1←14;
  ZTC1←15;
  ZFCO2←16;
  ZDTZJM←17;
  ZWW←18;
  ZNSTOP←19;
      OMIN←GRABONE(Z,ZOMIN);
      WW←GRABONE(Z,ZWW);
      DTZJM←GRABONE(Z,ZDTZJM);
      JSTAR←GRABONE(Z,ZJSTAR);
      PROB←GRABONE(Z,ZPROB);
      N←GRABONE(Z,ZN);
      TIME←GRABONE(Z,ZTIME);
      DTC←GRABONE(Z,ZDTC);
      UMIN←GRABONE(Z,ZUMIN);
      JFIN←GRABONE(Z,ZJFIN);
      CO2←GRABONE(Z,ZCO2);
      C1←GRABONE(Z,ZC1);
      TC1←GRABONE(Z,ZTC1);
```

```
        NSTOP←GRABONE(Z,ZNSTOP);
        FCO2←GRABONE(Z,ZFCO2);
 AGAIN: DELT←DTC ;
        DTC←DTZJM;
        DTZJM←100000.0;
        PMAX←0.0;
 SLEW(U,JMAX);SLEW(P,JMAX);SLEW(Q,JMAX);SLEW(RHO,JMAX);SLEW(X,JMAX);
SLEW(ZM,JMAX);SLEW(EPG,JMAX);SLEW(CS,JMAX);SLEW(T,JMAX);SLEW(RDRT,JMAX);
        JJ←JSTAR DIV 62;
    LOOP J←0.1,JJ DO
    BEGIN
    MODE←TRUE;
        P1←P[J]+Q[J];
    MODE←REVL(2×J,((PEN+(J×62))≤JSTAR)AND(NOT BOOLEAN(-1)));
 U[J]←U[J]+DELT×(P1-RTL(1,TRUE,P1))/(RHO[J]×(X[J]-RTL(-1,TRUE,X[J]))
    +RTL(1,TRUE,RHO[J])×(RTL(1,TRUE,X[J])-X[J]));
        IF(ABS(U[J])<UMIN) THEN U[J]←0.0;
        X[J]←X[J]+DTC×U[J];
    END;
    MODE←TRUE;
 ADJUSTSLEWED(U,JMAX);ADJUSTSLEWED(X,JMAX);
 J←JSTAR-64×JJ; IF(J<0) THEN J←J+64;
        IF(GRABONE(U[JJ],J)>0.) THEN JSTAR←JSTAR+1;
        IF(JSTAR≥JFIN) THEN GO TO STOP;
        JSTP1←JSTAR+1;
    LOOP J←0.1,JSTP1 DIV 62 DO
    BEGIN
    MODE←TRUE;
        P1←X[J];
        X3←P1×P1×P1;
    MODE←REVL(2×J,((PEN+(J×62))≤JSTP1)AND(NOT BOOLEAN(-1)));
        RHON←ZM[J]/(X3-RTL(-1,TRUE,X3));
        DV←(RHO[J]-RHON)/(RHON×RHO[J]);
        DU←(U[J]-RTL(-1,TRUE,U[J]));
        Q[J]←0.0;
        PDU←DU;
        IF(DU>0.0) THEN PDU←0.0;
        IF(DV<0.) THEN Q[J]←RHON×(PDU×CQ2-C1×CS[J])×PDU;
        IF(Q[J]≤QMIN) THEN Q[J]←0.0;
        CRNT←0.0;
        IF(Q[J]>0.0) THEN CRNT←TC1×CS[J]-FCO2×PDU;
        GMONE←0.4;
        P2←GMONE×RHON×EPG[J];
        E1←(EPG[J]-DV×P[J]);
        IF(E1<0.) THEN E1←0.;
        P1←GMONE×RHON×E1;
        EPG[J]←EPG[J]-(P2+P[J]+Q[J]+Q[J])×DV/(2.+(P2-P1)/P[J]);
        IF(EPG[J]<0.0) THEN EPG[J]←0.0;
        P[J]←GMONE×RHON×EPG[J];
        RHOF←RHON×1.001;
        IF(DV>0.0) THEN RHOF←RHON×.999;
        PFUDGE←GMONE×RHOF×EPG[J];
        CS[J]←SQRT((P[J]-PFUDGE)/(RHON-RHOF));
        RHO[J]←RHON;
        DTZJ←WW×(X[J]-RTL(-1,TRUE,X[J]))/(CRNT+.01+CS[J]);
        DTZJMR←MIN(DTZJ);
        PMAXR←MAX(P[J]);
    IF(DTZJMR<DTZJM) THEN DTZJM←DTZJMR;
    IF(PMAXR>PMAX) THEN PMAX←PMAXR;
    END;
 MODE←TRUE;
```

```
ADJUSTSLEWED(P,JMAX);ADJUSTSLEWED(Q,JMAX);ADJUSTSLEWED(CS,JMAX);
ADJUSTSLEWED(RHO,JMAX);ADJUSTSLEWED(EPG,JMAX);
      N←N+1;
      TIME←TIME+DTC;
      PUTCPR(PROB,Z,ZPROB);
      PUTCPR(DTZJM,Z,ZDTZJM);
      PUTCPR(PMAX,Z,ZPMAX);
      PUTCPR(TIME,Z,ZTIME);
      PUTCPR(DTC,Z,ZDTC);
      PUTCPI(N,Z,ZN);
      PUTCPI(JSTAR,Z,ZJSTAR);
      PUTCPR(DELT,Z,ZDELT);
      PUTCPR(XPMAX,Z,ZXPMAX);
      PUTCPI(JPMAX,Z,ZJPMAX);
      PUTCPI(JSTP1,Z,ZJSTP1);
      PUTCPI(NSTOP,Z,ZNSTOP);
 UNSLEW(U,JMAX);UNSLEW(P,JMAX);UNSLEW(Q,JMAX);UNSLEW(RHO,JMAX);
 UNSLEW(X,JMAX);UNSLEW(ZM,JMAX);UNSLEW(EPG,JMAX);UNSLEW(CS,JMAX);
 UNSLEW(T,JMAX);UNSLEW(RDRT,JMAX);
SIMWRITE(I4DISK1,Z);
      IF(N<NSTOP) THEN GO TO AGAIN;
STOP;
END.
```

APPENDIX VII

COMPUTER LISTING OF SAP IN ASK

```
%**************** ASK PROGRAM SAP ***************
 BEGIN
      CHWS    64;        %SELECT 64 BIT WORD MODE
%
%PREAL SUBROUTINE SORT AS RGA(PREAL X AS RGA)
%BEGIN
%
%         SORT ROUTINE FROM GLYPNIR
%
DEFINE COMMON=ASKRATCH##;
ASKRATCH[ENTRY]::
          BLK         14;
RASBS[ENTRY]::      WDS         64;
SORT:     FILL          ;
          EXCHL(1)   $D53;
          TXFFM(1)     ,2;
          SLTT(1)      =5;
          HALT          ;
          STORE(1)    $C3;
          EXCHL(1)   $D53;
%BEGIN
          JUMP      GSORT;
          FILL         16;
GSOT1:    DATA              (2.22152)1,(3.16214)1;
          DATA              (-2.03146)1,(-5.86118)1;
          DATA              (0.8125)1,(4.75)1;
          DATA              (40001000000000000000:8)1;
          FILL         16;
GSOT2:    STI(2)     $D58;
          RIN(1)     $D40;
          LIT(1) =1,47,40;
GSOT3:    LDI(2)     0(1);
          LDS         $C2;
          STS          *0;
          XI           =1;
          TXITAM(1) ,GSOT3;
          LDI(2)     $D58;
          EXCHL(2)   $ICR;
GSORT:    FILL           ;
          CLC(2)         ;
          SLTT(2)  =GSOT2;
          CLC(1)         ;
          SLTT(1)  =GSOT1;
          LDX =ASKRATCH+6;
          EXCHL(2)   $ICR;
          LDEE1       $C0;
          IB            0;
          SETI    I.AND.E;
          SETC(2)       I;
          ZERTA(2)     ,2;
          SLTT(1)     =12;    %TRIED TO FIND SORT(-N).
          HALT           ;
          JM7            ;
          SETC(1)       J;
          ONEST(1) ,GSOT4;
          LDX          =0;
          SETF   -J.AND.E;
          IB           15;
          SETC(1)       E;
          SETF;    E.OR.E;
          STA  ASKRATCH+5;
```

```
         SETF     I.AND.F;
         SETF1     E.OR.E;
         LDY           =1;
         ADFX ASKRATCH+12;
         SHAMR          1;
         STA   ASKRATCH+5;
         LDFF1        $C1;
         IB             1;
         SHAR           1;
         SETF     I.AND.F;
         SETF1     E.OR.E;
         SAR            1;
         RAR            2;
         LDFF1        $C1;
         SETF    -I.AND.E;
         SETF1     E.OR.E;
         SAR            2;
         LDFF1        $C1;
         STA   ASKRATCH+2;
         LDA   ASKRATCH+5;
         LEX  ASKRATCH+10;
         NORM            ;
         LIT(1)      =3.0;
         SAN             ;
         STA   ASKRATCH+3;
         MLRN *ASKRATCH+10;
         SBRN  *ASKRATCH+8;
         MLRN ASKRATCH+3;
         ADRN *ASKRATCH+6;
         STA   ASKRATCH+4;
         MLRN          $A;
         MLRN ASKRATCH+3;
         ADRN         $C1;
         MLRN ASKRATCH+4;
         SHAMR          1;
         NORM            ;
         STA   ASKRATCH+4;
         MLRN          $A;
         MLRN ASKRATCH+3;
         ADRN         $C1;
         MLRN ASKRATCH+4;
         SHAMR          1;
         NORM            ;
         STA   ASKRATCH+4;
         MLRN          $A;
         MLRN ASKRATCH+3;
         ADRN         $C1;
         MLRN ASKRATCH+4;
         SHAMR          1;
         NORM            ;
         SAN             ;
         MLRN ASKRATCH+3;
         ADFX ASKRATCH+2;
         NORM            ;
GSQTA;   LDFF1        $C0;
%END;
         EXCHL(1)    $D53;
         LOAD(1)      $C3;
         ALTT(1) =77777777(8);
         EXCHL(1)    $D53;
         STI(3)      $ICR;
```

```
          FILL
*
*    BEGIN SAP EXECUTION
*
SPEX: LIT(0)  =17777777777777777777777(8)
     STL(0)  $D33;      *STORE MODE TO ENABLE ALL PE'S TO $D33
     LDFE1   $C0;       *ENABLE ALL PE'S
     LIT(0)  =0.0;
     STL(0)  $D35;      *STORE 0.0 TO $D35
     LIT(0)  =1;
     STL(0)  $D36;      *STORE 1 TO $D36
        STL(0)    $D38;    *STORE WRITE RECORD COUNT
        LIT(0)  1,RASBS+64,RASBS;
        STL(0)    $D53;
*
*       SIMREAD(I4DISK2,7);
*
        CLC(3)          ;     *CLEAR ACAR3
        SLIT(3)      =44;     *WORD COUNT TO ACAR3
        CROTR(3)      14;     *ROTATE ACAR3 RIGHT 14 BITS
        CLC(2)          ;     *CLEAR ACAR2
        SLIT(2)       Z;      *Z ADDRESS TO ACAR2
        CSHR(2)       4;      *SHIFT ADDRESS RIGHT 4 BITS (DIVISABLE BY 16)
        COR(3)      $C2;      *ADDRESS TO ACAR3
        CROTR(3)     33;      *ROTATE ACAR3 RIGHT 33 BITS
                              *15 FOR ADDRESS
                              *2 FOR CONFIGURATION CONTROL
                              *16 FOR RECORD NUMBER
        LIT(2)       =2;      *FILE NUMBER TO ACAR2
        COR(3)      $C2;      *FILE NUMBER TO ACAR3
        CROTR(3)     17;      *ROTATE ACAR3 RIGHT 17 BITS
        CSR(3)        1;      *READ=1
        DISKIO         ;
*
*       SIMWRITE(I4DISK1,Z);
*
        CLC(3)          ;     *CLEAR ACAR3
        SLIT(3)      =44;     *WORD COUNT TO ACAR3
        CROTR(3)      14;     *ROTATE ACAR3 RIGHT 14 BITS
        CLC(2)          ;     *CLEAR ACAR2
        SLIT(2)      =Z;      *Z ADDRESS TO ACAR2
        CSHR(2)       4;      *SHIFT ADDRESS RIGHT 4 BITS (DIVISABLE BY 16)
        COR(3)      $C2;      *ADDRESS TO ACAR3
        CROTR(3)     33;      *ROTATE ACAR3 RIGHT 33 BITS
                              *15 FOR ADDRESS
                              *2 FOR CONFIGURATION CONTROL
                              *16 FOR RECORD NUMBER
        LIT(2)       =1;      *FILE NUMBER TO ACAR2
        COR(3)      $C2;      *FILE NUMBER TO ACAR3
        CROTR(3)     17;      *ROTATE ACAR3 RIGHT 17 BITS
        CRR(3)        1;      *WRITE=0
        DISKIO         ;
*
* STORE PE 7 CONSTANTS TO ADR ($D0-$D31);
*
     LDA     7;          *LOAD Z TO RGA
     CLC(1)   ;
     CSR(1)  0;         *PE ENABLE MODE FOR INDEX LOOP
     LIT(0)  1,31,0;     *LOOP INDEX
       CLC(3)        ;    *CLEAR ACAR3 FOR INDEX
     LDFE1   $C1;        *ENABLE INDEX PE
```

```
        LDC(2)  $A;         %Z IN ENABLED PE TO ACAR2
        STL(2)  $D0(3));    %STORE Z
          CADD(3)     $D36;    %INCREMENT ADDRESS INDEX
        CSHR(1) 1;          %ENABLE NEXT PE
          TXITM(0)    ,-6;     %LOOP ON INDEX
        LDL(0)  $D33;
        LDFE1   $C0;        %ENABLE ALL PE'S
%
%          DTZJ+100000.0;
%
           LIT(1) =100000.0;   %
           LDA          $C1;   %
           STA         DTZJ;   %STORE DTZJ
%
      LIT(1)     =63;        %63 TO ACAR1
      CSUB(1)    $D5;        %63-JSTART TO ACAR1
      STL(1)     $D37;       %SAVE 63-JSTAR IN $D37
%AGAIN:  DELT+DTC;
%
 AGAIN:LDL(0) $D4;          %DTC TO ACAR0
        STL(0)  $D3;        %STORE DELT
%
%          DTC+DTZJM;
%
        LDL(0)  $D17;       %DTZJM TO ACAR0
        STL(0)  $D4;        %STORE DTC
%
%          MODE+(PEN≤JSTAR) AND (NOT BOOLEAN(-0));
%
      LDL(1)     $D33;       %SET ALL BITS IN ACAR1
      LDL(2)     $D5;        %LOAD JSTAR
      CADD(2)    $D36;       %JSTAR + 1 TO ACAR2
      CSHR(1)   0(2);        %SHIFT ACAR1 RIGHT, (JSTAR+1) BITS
      COMPC(1)   ;           %COMPLEMENT ACAR1
      CRB(1)     0;          %CLEAR SIGN BIT
      STL(1)     $D34;       %SAVE MODE IN $D34
           LDFE1        $C1;   %SET PE MODE
%
%          U+U+DELT×((P+Q)-RTL(1,TRUE,(P+Q)))/(RHO×(X-RTL(-1,TRUE,X))
%          &RTL(1,TRUE,RHO)×(RTL(1,TRUE,X)-X)));
%
           LDL(0)       $D33;  %
           LDFE1         $C0;  %ENABLE ALL PE'S
           LDA            X;   %LOAD X TO RGA
           LDS           $A;   %SAVE X IN RGS
           RTL          $A,1;  %RTL(-1,TRUE,X) TO RGR
           SBRN          $R;   %X-RTL(-1,TRUE,X) TO RGA
           MLRN         RHO;   %RHO×(X-RTL(-1,TRUE,X)) TO RGA
           LDB           $A;   %SAVE ABOVE IN RGB
           LDA           $S;   %X FROM RGS TO RGA
           LDS           $B;   %RGB TO RGS
           RTL         $A,63;  %RTL(1,TRUE,X) TO RGR
           CHSA           ;    %-X TO RGA
           ADRN          $R;   %RTL(1,TRUE,X)-X TO RGA
           LDB           $A;   %SAVE ABOVE IN RGB
           LDA          RHO;   %LOAD RHO TO RGA
           RTL         $A,63;  %RTL(1,TRUE,RHO) TO RGR
           LDA           $R;   %RTL(1,TRUE,RHO) TO RGA
           MLRN          $R;   %RTL(1,TRUE,RHO)×(RTL(1,TRUE,X)-X) TO RGA
           ADRN          $S;   %DENOMINATOR TO RGA
           LDS           $A;   %SAVE DENOMINATOR TO RGS
```

```
          LDA            P;     %LOAD P TO RGA
          ADRN           Q;     %P+Q TO RGA
          RTL        $A,63;     %RTL(1,TRUE,(P&Q)) TO RGA
          SBRN          $R;     %(P&Q)-RGA TO RGA
          LDL(0)      $D34;     %
          LDFF1        $C0;     %RESTORE EXECUTION MODE
          LDR           =0;
          DVRN          $S;     %(.....) TO RGA
          LDS           $A;     %SAVE (.....)
          LDL(1)       $D3;     %
          LDA          $C1;     %DELT TO ACAR1
          MLRN          $S;     %DELT×(.....)
          ADRN           U;     %UPDATE U
          STA            U;     %STORE U
%
%         IF(ABS(U)<UMIN) THEN U+0.;
%
%U IN RGA FROM PREVIOUS STATEMENT
%
      RAB      =0;          %ABS(U) TO RGA
          LDL(2)       $D8;     %UMIN TO ACAR2
      IAL      $C2;         %SET PE MODE REGISTER I IF ABS(U)<UMIN
      SETC(2)  T;           %PE I MODE BIT PATTERN TO ACAR2
      CRB(2)   0;           %CLEAR ACAR2 SIGN BIT
          LDL(3)      $D37;
          CSHR(2)     0(3);
          CSHL(2)     0(3);
          LDFF1        $C2;     %ENABLE PE'S FOR WHICH ABS(U)<UMIN
          LDL(3)      $D35;     %LOAD ACAR3 WITH 0.0
          LDA          $C3;     %
          STA            U;     %STORE U=0.0
          LDL(3)      $D34;     %
          LDFF1        $C3;     %STORE CURRENT EXECUTION MODE
%
%     X+X+DTC×U
%
      LDL(1)   $D4;         %DTC TO ACAR1
      LDA      U;           %U TO RGA
      MLRN     $C1;         %DTC×U TO RGA
      ADRN     X;           %X+DTC×U TO RGA
      STA      X;           %STORE RGA TO X
%
%     IF(GRABONE(U,JSTAR)>0.) THEN JSTAR=JSTAR+1;
%
      CLC(0)   ;            %CLEAR ACAR0
      SLIT(0)  =U;          %PE ADDRESS OF U TO ACAR0
      CADD(0)  $D5;         %ADDRESS OF U IN JSTAR PE
      LOAD(0)  $C1;         %LOAD U(JSTAR) TO ACAR1
      STL(1)   $C2;         %SAVE U(JSTAR) IN ACAR2
      LIT(3)   =63;         %SHIFT COUNT TO ACAR3
          CSHR(1)     0(3);
      ZERT(1)  ,1;          %SKIP IF U(JSTAR) ≥ 0
      SKIP     ,6;          %SKIP IF U(JSTAR) < 0
      ZERT(2)  ,5;          %SKIP IF U(JSTAR) = 0
      LDL(0)   $D36;        %U(JSTAR) > 0.0
      CADD(0)  $D5;
      STL(0)   $D5;         %STORE JSTAR
          CSUB(3)       $D5;    %
          STL(3)      $D37;     % UPDATA 63-JSTAR
%
%     IF (JSTAR≥JFIN) THEN GO TO STOP;
```

```
*
     LDL(1)      $D9;        %JFIN TO ACAR1
     CSUB(1)     $D5;        %JFIN-JSTAR TO ACAR1
     ZERXT(1)    ,3;         %SKIP TO HALT
     STL(1)      $C2;        %SAVE ACAR1 IN ACAR2
     CSHR(1)     23;         %SHIFT 24 BIT WORD SIGN BIT
     ZERXT(1)    ,1;         %SKIP IF JSTAR < JFIN
     HALT        ;           %STOP
*
*    JSTP1 ← JSTAR+1
*
     LDL(1)      $D36;       %SET ACAR1 TO 1
     CADD(1)     $D5;        %JSTAR+1 TO ACAR1
     STL(1)      $D6;        %STORE JSTAR+1 TO JSTP1
*
*    MODE ← SHIFTR(1,MODE OR BOOLEAN (=0));
*
     LDL(0)      $D34;       %LOAD CURRENT EXECUTION MODE TO ACARO
     CSB(0)      0;          %SET ACARO SIGN BIT
     CSHR(0)     1;          %SHIFT ACARO 1 BIT RIGHT
     LDEE1       $C0;        %STORE MODE TO PE S
     STL(0)      $D34;       %STORE MODE
*
*    X3 ← X×X×X
*
         LDL(0)        $D33;     %
         LDEE1         $C0;      %ENABLE ALL PE'S
     LDA         X;          %X TO RGA
     LDS         $A;         %SAVE X IN RGS
         MLRN          $S;       %X×X TO PE RGA'S
     MLRN        $S;         %X3= X×X×X TO RGA
*
*    RHON ← ZM/(X3-RTL(-1,TRUE,X3))
*
     RTL         $A,1;       %ROUTE X3 LEFT-1
     SBRN        $R;         %X3-RTL(-1,TRUE,X3)) TO RGA
     LDS         $A;         %X3-RTL(-1,TRUE,X3)) TO RGS
         LDL(0)        $D34;     %
         LDEE1         $C0;      %RESTORE EXECUTION MODE
     LDA         ZM;         %ZM TO RGA
         LDR           =0;       %CLEAR PE RGB'S
     DVRN        $S;         %ZM/(X3-RTL(-1,TRUE,X3)) TO RGA
     STA         RHON;       %(RGA) TO RHON
*
*    DV ← (RHO-RHON)/(RHON×RHO)
*
     LDS         $A;         %RHON TO RGS
     LDA         RHO;        %RHO TO RGA
     LDR         $A;         %SAVE RHO TO RGR
     MLRN        $S;         %RHO*RHON TO RGA
         LDR           $A;       %SAVE RHO×RHON TO RGR
         LDA           $R;       %RHO TO RGA
         LDR           $R;       %SAVE RHO×RHON IN RGR
     SBRN        $S;         %RHO-RHON TO RGA
         LDR           =0;       %CLEAR PE RGB'S
     DVRN        $R;         %(RHO-RHON)/(RHON×RHO) TO RGA
     STA         DLV;        %STORE(RHO-RHON)/RHO×RHO) TO DLV
*
*    DU ← (U-RTL(-1,TRUE,U));
*
         LDL(0)        $D33;     %
```

```
        LDFF1       $C0;     %ENABLE ALL PE'S
    LDA         U;        %LOAD RGA WITH U
    RTL          $A,1;    %RTL(-1,TRUE,U) TO RGR
    SBRN         $R;      %
        LDI(0)      $D34;    %
        LDFF1        $C0;    %RESTORE EXECUTION MODE
    STA         DU;       %DU=(U-RTL(-1,TRUE,U)) TO DU
        LDS          $A;     %SAVE DU IN PE RGS'S
%
%   Q + 0.0;
%
    LIT(1)     =0.0;      %0.0 TO ACAR1
    LDA        $C1;       %ACAR1 TO RGA
    STA        Q;         %(RGA=0.0) TO Q
%
%       IF(DU>0.0 THEN PDU+0.0 ELSE PDU+DU
%
        LDA          DU;     %DU TO RGA
        LDI(1)      $D35;    %0.0 TO ACAR1
        IAG          $C1;    %SET PE I MODE BIT IF DU>0.0
        SETC(2)       I;     %PE I MODE BIT PATTERN TO ACAR2
        CRB(2)        0;     %CLEAR PE 0 MODE BIT
        LDI(3)      $D37;
        CSHR(2)     0(3);
        CSHL(2)     0(3);
        LDFF1        $C2;    %ENABLE PE'S FOR WHICH DU>0.0
        LDI(3)      $D35;    %LOAD ACAR3 WITH 0.0
        LDA          $C3;    %0.0 TO ENABLED PE'S
        LDI(3)      $D34;    %
        LDFF1        $C3;    %STORE CURRENT EXECUTION MODE
        STA          PDU;    %STORE PDU
%
%       IF(DV≤0.) THEN Q+RHONX(PDU×CD2-C1×CS)×PDU;
%
        LDI(1)      $D35;    %0.0 TO ACAR1
        LDA          DLV;    %LOAD RGA WITH DLV
        IAG          $C1;    %SET RGD I BIT IF DLV>0.0
        SETC(1)       I;     %PE RGD S TO ACAR1
        CRB(1)        0;     %CLEAR ACAR1 SIGN BIT
        LDI(3)      $D37;
        CSHR(1)     0(3);
        CSHL(1)     0(3);
        COMPC(1)       ;     %COMPLEMENT ACAR1
        CRB(1)        0;     %CLEAR ACAR1 SIGN BIT
        CSHR(1)     0(3);
        CSHL(1)     0(3);
        LDFF1        $C1;    %ENABLE PE'S FOR WHICH DV<0.
        LDA         CS;      %LOAD RGA WITH CS
        LDI(1)      $D14;    %LOAD C1 TO ACAR1
        MLRN         $C1;    %C1×CS TO RGA
        LDS          $A;     %SAVE C1×CS IN RGS
        LDA          PDU;    %LOAD RGA WITH PDU
        LDR          $A;     %SAVE PDU IN RGR
        LDI(1)      $D13;    %LOAD CD2 TO ACAR1
        MLRN         $C1;    %PDU×CD2 TO RGA
        SBRN         $S;     %(PDU×CD2-C1×CS) TO RGA
        MLRN         $R;     %(PDU×CD2-C1×CS)×PDU TO RGA
        MLRN         RHON;   %RHON×(PDU×CD2-C1×CS)×PDU TO RGA
        STA           Q;     %STORE Q
        LDI(0)      $D34;    %
        LDFF1        $C0;    %RESTORE CURRENT EXECUTION MODE
```

```
%
%        IF(Q≤QMIN) THEN Q←0.;
%
         LDA          Q;     %LOAD RGA WITH Q
         LDL(1)     $D7;     %LOAD QMIN TO ACAR1
         IAG        $C1;     %SET RGD I BIT IF Q>QMIN
         SETC(1)      I;     %PE RGD S TO ACAR1
         CRB(1)       0;     %CLEAR ACAR1 SIGN BIT
         LDL(3)    $D37;
         CSHR(1)   0(3);
         CSHL(1)   0(3);
         COMPC(1)     ;      %COMPLEMENT ACAR1
         CRB(1)       0;     %CLEAR ACAR1 SIGN BIT
         CSHR(1)   0(3);
         CSHL(1)   0(3);
         LDEE1      $C1;     %ENABLE PE'S FOR WHICH Q≤QMIN
         LDL(1)    $D35;     %LOAD 0.0 TO ACAR1
         LDA        $C1;
         STA          Q;     %STORE Q
         LDL(0)    $D34;     %
         LDEE1      $C0;     %RESTORE CURRENT EXECUTION MODE
%
%        CRNT←0.0;
%        IF(Q>0.0) THEN CRNT←TC1×CS-FCO2×PDU;
%
%Q IN RGA FROM PREVIOUS STATEMENT
%
         LDL(1)    $D35;     %
         LDA        $C1;     %
         STA       CRNT;     %STORE CRNT
         LDA          Q;     %LOAD Q
         IAG        $C1;     %SET RGD I BIT IF Q>0.0
         SETC(1)      I;     %PE RGD S TO ACAR1
         CRB(1)       0;     %CLEAR ACAR1 SIGN BIT
         LDL(3)    $D37;
         CSHR(1)   0(3);
         CSHL(1)   0(3);
         LDEE1      $C1;     %ENABLE PE'S FOR WHICH Q>0.0
         LDA        PDU;     %LOAD RGA WITH PDU
         LDL(1)    $D16;     %FCO2 TO ACAR1
         MLRN       $C1;     %FCO2×PDU TO RGA
         LDS         $A;     %SAVE FCO2×PDU IN RGS
         LDA         CS;     %LOAD RGA WITH CS
         LDL(1)    $D15;     %LOAD TC1 TO ACAR1
         MLRN       $C1;     %TC1×CS TO RGA
         SBRN        $S;     %TC1×CS-FCO2×PDU TO RGA
         STA       CRNT;     %STORE CRNT
         LDL(0)    $D34;
         LDEE1      $C0;     %RESTORE CURRENT EXECUTION MODE
%
%    GMONE ← 0.4;
%
     LIT(1)    =0.4;      %SET ACAR1 TO 0.4
     LDA       $C1;       %LOAD PE RGA S WITH 0.4
     STA       GMONE;     %STORE 0.4 TO GMONE
%
%    P2 ← GMONE×RHON×EPG;
%
     MLRN      RHON;      %GMON×RHON TO PE RGA S
     LDS       EPG;       %LOAD PE RGS S WITH EPG
     MLRN      $S;        %GMONE×RHON×EPG TO PE RGA S
```

```
        STA       P2;       %STORE GMONE×RHON×EPG TO P2
%
%       E1 ← (EPG-DV×P);
%
            LDA        DLV;    %DLV TO PE RGA'S
        CHSA      ;         %-DV TO PE RGA S
        MLRN      P;        %-DV×P TO PE RGA S
        ADRN      $S;       %(EPG-DV×P) TO PE RGA S
%
%           IF(E1<0.) THEN E1←0.;
%
            LDI(1)     $D35;    %LOAD 0.0 TO ACAR1
            IAI          $C1;   %SET RGD I BIT IF E1<0.
            SETC(1)        I;   %PE RGD S TO ACAR1
            CRB(1)         0;   %CLEAR ACAR1 SIGN BIT
            LDI(3)     $D37;
            CSHR(1)    0(3);
            CSHL(1)    0(3);
            LDEE1        $C1;   %ENABLE PE'S FOR WHICH E1<0.0
            LDI(3)     $D35;    %LOAD ACAR3 WITH 0.0
            LDA           $C3;  %0.0 TO ENABLED PE'S
            LDI(3)     $D34;    %
            LDEE1        $C3;   %RESTORE CURRENT EXECUTION MODE
            STA           E1;   %STORE E1
%
%       P1 ← GMONE×RHON×E1
%
        MLRN      RHON;     %RHON×E1 TO PE RGA S
        MLRN      GMONE;    %GMONE×RHON×E1 TO PE RGA S
        STA       P1;       %STORE GMONE×RHON×E1 TO P1
%
%       EPG ← EPG-((P2+P+Q+Q)×DV/(2.+(P2-P1)/P));
%
        CHSA      ;         %-P1 TO PE RGA S
        ADRN      P2;       %P2-P1 TO PE RGA S
        LDB       =0;       %CLEAR PE RGB S
            LDR             P;
        DVRN      $R;       %(P2-P1)/P TO PE RGA S
        LDS       $A;       %(P2-P1)/P TO PE RGS S
        LIT(1)    =2.0;     %SET ACAR1 TO 2.0
        LDA       $C1;      %LOAD PE RGA S WITH 2.0
        ADRN      $S;       %(2.×(P2-P1)/P) TO PE RGA S
        STA       TMP1;     %STORE DIVISOR=(2.+(P2-P1)/P) TO TMP1
        LDA       Q;        %Q TO PE RGA S
        LDS       $A;       %SAVE Q IN PE RGS S
        ADRN      $R;       %P+Q TO PE RGA S
        ADRN      $S;       %P+Q+Q TO PE RGA S
        ADRN      P2;       %P2+P+Q+QTO PE RGA S
        MLRN      DLV;      %(P2+P+Q+Q)×DLV TO PE RGA'S
        LDB       =0;       %CLEAR PE RGB S
        DVRN      TMP1;     %STORE(P2+P+Q+Q)×DV/(2.0+(P2-P1)/P) TO PE RGA S
        CHSA      ;         %CHANGE PE RGA SIGNS
        ADRN      EPG;      %EPG-(P2+P+Q+Q)×DV/(2.0+(P2-P1)/P) TO PE RGA S
%
%           IF(EPG<0.) THEN EPG←0.;
%
            LDI(1)     $D35;    %LOAD 0.0 TO ACAR1
            IAI          $C1;   %SET RGD I BIT IF EPG<0.
            SETC(1)        I;   %PE RGD S TO ACAR1
            CRB(1)         0;   %CLEAR ACAR1 SIGN BIT
            LDI(3)     $D37;
```

```
        CSHR(1)      0(3);
        CSHL(1)      0(3);
        LOFF1          $C1;     %ENABLE PE'S FOR WHICH EPG<0.0
        LDI(3)       $D35;     %LOAD ACAR3 WITH 0.0
        LDA            $C3;     %0.0 TO ENABLED PE'S
        LDI(3)       $D34;     %
        LDFF1          $C3;     %RESTORE CURRENT EXECUTION MODE
        STA           EPG;     %STORE EPG
%
%    P ← GMONE×RHON×EPG;
%
    LDA         RHON;      %RHON TO PE RGA S
    LDS         $A;        %SAVE RHON IS PE RGS S
    MLRN        GMONE;     %GMONE×RHON TO PE RGA S
    MLRN        EPG;       %GMONE×RHON×EPG TO PE RGA S
    STA         P;         %STORE GMONE×RHON×EPG TO P
%
%    RHOF ← RHON×1.001;
%
    LIT(1)      =1.001;    %SET ACAR1 TO 1.001
    LDA         $C1;       %LOAD PE RGA S WITH 1.001
    MLRN        $S;        %RHON×1.001 TO PE RGA S
    STA         RHOF;      %STORE RHON×1.001 TO RHOF
%
%        IF(DV≥0.0) THEN RHOF←RHON×.999;
%
        LDA           DLV;     %LOAD RGA WITH DLV
        LDI(1)       $D35;     %LOAD 0.0 TO ACAR1
        IAI           $C1;     %SET RGD I BIT IF DLV<0.0
        SETC(1)         I;     %PE RGD S TO ACAR1
        CRB(1)          0;     %CLEAR ACAR1 SIGN BIT
        LDI(3)       $D37;
        CSHR(1)      0(3);
        CSHL(1)      0(3);
        COMPC(1)         ;     %COMPLEMENT ACAR1
        CRB(1)          0;     %CLEAR ACAR1 SIGN BIT
        CSHR(1)      0(3);
        CSHL(1)      0(3);
        LOFF1         $C1;     %ENABLE PE'S FOR WHICH DV≥0.0
        LIT(1)      =.999;     %
        LDA           $C1;     %0.999 TO RGA
        MLRN         RHON;     %RHON×.999 TO RGA
        LDI(0)       $D34;     %
        LDFF1         $C0;     %RESTORE CURRENT EXECUTION MODE
        STA          RHOF;     %STORE RHOF
%
%    PFUDGE ← GMONE×RHOF×EPG;
%
    LDA         RHOF;      %RHOF TO PE RGA S
    LDS         $A;        %SAVE RHOF IN PE RGS S
        MLRN        GMONE;     %GMONE×RHOF TO PE RGA'S
    MLRN        EPG;       %GMONE×RHOF×EPG TO PE RGA S
    STA         PFUDGE;    %STORE GMONE×RHOF×EPG TO PFUDGE
%
%    CS ← SQRT((P-PFUDGE)/(RHON-RHOF));
%
    CHSA        ;          %-PFUDGE TO PE RGA S
    ADRN        P;         %P-PFUDGE TO PE RGA S
    LDB         $A;        %SAVE P-PFUDGE IN PE RGB S
    LDA         $S;        %RHOF TO PE RGA S
        LDS          $B;       %P-PFUDGE TO PE RGS'S
```

```
        CHSA      ;         %-RHOF TO PE RGA S
        ADRN      RHON;     %RHON-RHOF TO PE RGA S
        LOB       $A;       %SAVE RHON-RHOF IN PE RGB S
        LDA       $S;       %P-PFUDGE TO PE RGA S
            LDS          $B;     %SAVE RHON-RHOF IN PE RGS:S
        LOB       =0;       %CLEAR PE RGB S
        DVRN      $S;       %(P-PFUDGE)/(RHON-RHOF) TO PE RGA S
            LDI(0)       $D34;   %CURRENT PE EXECUTION MODE TO ACAR0
        CLC(3)    ;
        SLIT(3)   SORT;
        EXCHL(3)  $ICR;     %JUMP TO SORT
        STA       CS;       %STORE SORT((P-PFUDGE)/(RHON-RHOF)) TO CS
%
%       DTZJ + WWX(X-RTL(-1,TRUE,X))/(CRNT+0.1+CS));
%
        LIT(1)    =0.01;    %SET ACAR1 TO 0.01
        ADRN      $C1;      %0.1+CS TO PE RGA S
        ADRN      CRNT;     %CRNT+0.1+CS TO PE RGA S
        LDS       $A;       %SAVE(CRNT+0.1+CS) IN PE RGS S
            LDI(0)       $D33;   %
            LDEE1        $C0;    %ENABLE ALL PE:S
        LDA       X;        %X TO PE RGA S
        RTL        $A,1;    %ROUTE X LEFT -1
        SBRN      $R;       %(X-RTL(-1,TRUE,X)) TO PE RGA S
            LDI(1)       $D18;   %LOAD WW TO ACAR1
        MLRN      $C1;
            LDI(0)       $D34;   %
            LDEE1        $C0;    %RESTORE EXECUTION MODE
            LDB          =0;     %CLEAR PE RGB:S
            DVRN         $S;     %WWx(X-RTL(-1,TRUE,X))O(CRNT&0.1&CS) TO RGA
        STA       DTZJ;
%
%           DTZJM+MIN(DTZJ);
%
            LDA          DTZJ;   %LOAD DTZJ
            LDI(0)       $D34;   %LOAD CURRENT EXECUTION MODE TO ACAR0
            COMPC(0)        ;    %COMPLEMENT MODE
            CLC(1)          ;    %
            COMPC(1)        ;    %
            CCB(1)         0;    %LARGEST POSITIVE NUMBER TO ACAR1
            LDEE1        $C0;    %
            LDA          $C1;    %LOAD THIS NUMBER TO DISABLED PE S
            COMPC(0)        ;    %RESTORE EXECUTION MODE
            LIT(1)     1,6,1;    %LOOP PARAMETERS TO ACAR1
            CLC(2)          ;    %CLEAR ACAR2
            SLIT(2)        1;    %ACAR2 SET TO 1
            SETF     E OR -E;    %
            SETF1   E1 OR E;     %
            RTL      $A,0(2);    %
            IAG           $R;    %
            SETF     I AND E;    %
            SETF1  I AND E1;     %
            LDA           $R;    %
            CSHL(2)        1;    %
            TXITAM(1)     ,-9;   %LOOP TIL MINIMUM
            LDEE1        $C0;    %RESTORE EXECUTION MODE
            LDC(1)       $A;     %MINIMUM IN RGA TO ACAR1
            STI(1)       $D17;   %STORE DTZJM
%
%           RHO+RHON
%
```

```
        LDA          RHON;    %LOAD RHON
        STA           RHO;    %STORE RHO
%
%       PMAX+MAX(P);
%
        LDA             P;    %LOAD P
        LDI(0)      $D34;     %LOAD CURRENT EXECUTION MODE TO ACAR0
        COMPC(0)        ;     %COMPLEMENT MODE
        CLC(1)          ;     %
        COMPC(1)        ;     %LARGEST NEGATIVE NUMBER TO ACAR1
        LDEE1        $C0;     %
        LDA          $C1;     %LOAD THIS NUMBER TO DISABLED PE S
        COMPC(0)        ;     %RESTORE EXECUTION MODE
        LIT(1)     1,6,1;     %LOOP PARAMETERS TO ACAR1
        CLC(2)          ;     %CLEAR ACAR2
        SLIT(2)        1;     %ACAR2 SET TO 1
        SETE     E OR -E;     %
        SETE1   E1 OR E;      %
        RTL      $A,0(2);     %
        IAL           $R;     %
        SETE     I AND E;     %
        SETE1   I AND E1;     %
        LDA           $R;     %
        CSHL(2)        1;     %
        TXLTAM(1)    ,-9;     %LOOP TIL MAXIMUM
        LDEE1        $C0;     %RESTORE EXECUTION MODE
        LDC(1)        $A;     %MAXIMUM IN RGA TO ACAR1
        STL(1)      $D10;     %STORE PMAX
%
%       TIME+TIME+DTC;
%
        LDL(1)       $D2;     %LOAD TIME TO ACAR1
        LDA          $C1;     %TIME TO RGA
        LDL(1)       $D4;     %LOAD DTC TO ACAR1
        ADRN         $C1;     %TIME+DTC TO RGA
        LDC(1)        $A;     %TIME+DTC TO ACAR1
        STL(1)       $D2;     %STORE TIME
%
%       STORE ADR CONSTANTS ($D0-$D31) TO PE Z ARRAY
%
        CLC(1)          ;     %CLEAR ACAR1
        CSR(1)         0;     %PE ENABLE MODE FOR INDEX LOOP
        LIT(0)    1,31,0;     %LOOP INDEX
        CLC(3)          ;     %CLEAR ACAR3 FOR INDEX
LP2:    LDL(2)    $D0(3);     %LOAD ADR TO ACAR2
        LDEE1        $C1;     %ENABLE INDEX PE
        LDA          $C2;     %ADR TO PE RGA
        CADD(3)     $D36;     %INCREMENT ADDRESS INDEX
        CSHR(1)        1;     %ENABLE NEXT PE
        TXLTM(0)    ,LP2;     %LOOP ON INDEX
        LDL(0)      $D34;     %
        LDEE1        $C0;     %ENABLE ALL PE'S
        STA            Z;     %STORE Z
%
%       MODE+TRUE;
%
        LDL(1)      $D33;     %LOAD TRUE TO ACAR1
        LDEE1        $C1;     %SET MODE
        STL(1)      $D34;     %STORE TO CURRENT EXECUTION MODE
%
%       SIMWRITE(IADISK1,Z);
```

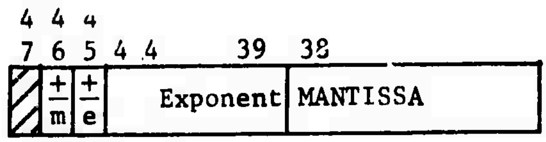
```
*
        CLC(3)          ;       *CLEAR ACAR3
        SLIT(3)      =44;       *WORD COUNT TO ACAR3
        CROTR(3)      14;       *ROTATE ACAR3 RIGHT 14 BITS
        CLC(2)          ;       *CLEAR ACAR2
        SLIT(2)       =Z;       *Z ADDRESS TO ACAR2
        CSHR(2)        4;       *SHIFT ADDRESS RIGHT 4 BITS (DIVISABLE BY 16)
        COR(3)       $C2;       *ADDRESS TO ACAR3
        CROTR(3)      15;       *ADDRESS
        CROTR(3)       6;       *CONFIGURATION
        LDI(2)      $D38;
        COR(3)       $C2;
        CROTR(3)      12;       *RECORD NUMBER
        LIT(2)        =1;       *FILE NUMBER TO ACAR2
        COR(3)       $C2;       *FILE NUMBER TO ACAR3
        CROTR(3)      17;       *ROTATE ACAR3 RIGHT 17 BITS
        CRR(3)         1;       *WRITE=0
        DISKIO          ;
        LDI(2)      $D38;       *
        CADD(2)     $D36;       *
        STI(2)      $D38;       *UPDATE RECORD COUNT
*
*       N+N+1
*
        LDL(1)       $D1;       *LOAD N TO ACAR1
        CADD(1)     $D36;       *N+1 TO ACAR1
        STI(1)       $D1;       *STORE N
*
*       IF(N<NSTOP) THEN GO TO AGAIN;
*
        LDI(1)      $D19;       *NSTOP TO ACAR1
        CSUB(1)      $D1;       *NSTOP-N TO ACAR1
        ZERT(1)       ,2;       *SKIP IF NSTOP-N=0
        CSHR(1)       23;       * SHIFT ACAR1 23 BITS RIGHT
        ZERT(1)       ,1;       *SKIP IF NSTOP-N<0
        HALT            ;       *NSTOP-N≤0
        JUMP      AGAIN;        *NSTOP-N<0
 Z:     BLK       1;
 U:     BLK       1;
 P:     BLK       1;
 O:     BLK       1;
 RHO:   BLK       1;
 X:     BLK       1;
 ZM:    BLK       1;
 EPG:   BLK       1;
 CS:    BLK       1;
 T:     BLK       1;
 RDRT:BLK         1;
 X3:    BLK       1;
 RHON:BLK         1;
 DLV:   BLK       1;
 CRNT:BLK         1;
GMONE:BLK         1;
P2:     BLK       1;
 F1:    BLK       1;
 P1:    BLK       1;
 PDU:   BLK       1;
 RHOF:BLK         1;
 DTZJ:BLK         1;
 DU:    BLK       1;
 PFUDGE:BLK       1;
 TMP1:BLK 1;
 END SPEX.
```

APPENDIX VIII

B5500/6500, ILLIAC IV, AND CDC 6600 WORD FORMATS

The B5500 and B6500 have the same 48-bit word formats because the machines were intended to be compatible. The difference in the description of these two word formats is that the high order bit in the B5500 is labeled bit zero and the low order bit is thus bit 47; in the B6500 the low order bit is numbered zero and the high order bit is numbered 47. Bits that can be manipulated by the system have been added outside the left-hand end of the word, so these added bits become 48 through 50. This distinction is relevant only when the user employs such functions as cocatinate (similar to INBY on the CDC 6600) where one references bits within the word.

1. B6500 WORD FORMAT



Floating point

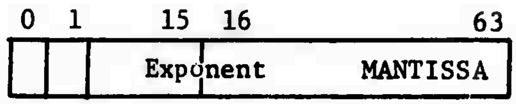
Integer

The B6500 floating-point number, or integer, contains a 39-bit mantissa in bits 0 to 39; a sign bit for this mantissa is contained in bit 46, word format use bit 47. For the floating-point word format an octal (SIC) exponent is contained in bits 39 to 44 and an exponent sign bit is contained in bit 45. Note that this is not the normal biased exponent that is used in many other machines.

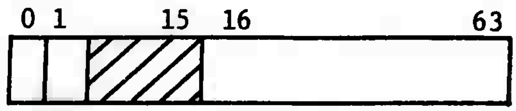
In the integer format the fields used by the exponent and its sign in the floating-point format are zero. In the integer format the largest integer that can be held in this format is $2^{39}-1$. Beyond this the hardware will convert the result of an arithmetic operation to floating-point format. Because the binary point is assumed to be to the right of bit zero, this conversion from integer to real formats is a continuous one--an integer appears to be just an "unnormalized" floating-point number with zero exponent. The value of a floating point A is calculated from the mantissa M and exponent E as

$$A = \pm M \times 8^{\pm E}$$

2. ILLIAC IV WORD FORMAT



Floating Point



Integer

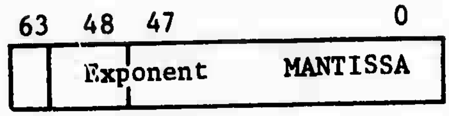
The ILLIAC IV has a 64-bit word that can contain either 48-bit integers or a floating-point word with a 48-bit mantissa. The binary point for the floating-point numbers is assumed to be at the high order end of the mantissa, that is, between bits 15 and 16 in the above figure. A sign bit for either the mantissa or for integers is contained in the high order bit, bit 0. For integers, bits 1 to 15 are zero. For floating-point numbers, this field contains a 15-bit exponent. This exponent is biased at 40000_8 and thus alleviates the need for an exponent sign bit. The value of a floating-point number A is calculated from the mantissa M, the exponent E, and the sign bit S as

$$A = (-1)^S \times 2^{(E-40000_8)}$$

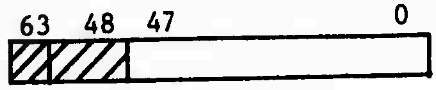
It is noted that a 32-bit half-precision mode also exists in the ILLIAC. Because GLYPNIR does not support this mode at this time, the word format for the 32-bit half-precision mode will not be described.

3. THE CDC 6600 WORD FORMAT

The format of the CDC 6600 integer and floating-point words are included here because we feel that any calculations done on the ILLIAC will be compared at some time with results arrived at on a CDC 6000 or 7000 series machine. Fortunately, the mantissa in both of these word formats is the same--48 bits. The difference in word length is all absorbed in the exponent field. (The extra bits are needed on the ILLIAC so it can have one bit per PE for logical instructions.)



Floating Point



Integer

The CDC 6000 series machines are ones-compliment machines, thus there is no sign bit and the negative of a number is its ones-compliment. Integers must be less than 2^{48} in size so that they can be processed with the same hardware that processes the 48-bit mantissa of a floating-point number. Although the bits 48 to 63 are not used (i.e., are zero) for positive integers, there is sign extension through this region for negative numbers, and thus they are all on in this case. In the floating-point format the binary point is assumed to be a the far right of the mantissa, and the exponent in bits 48 to 62 is biased at 2000_8.

Because the mantissa of the CDC 6600 word and that of the ILLIAC are of the same length, the movement of algorithms from one machine to the other should be simplified, as should the movement of data and calculations. The movement of data for later calculation between machines of different word (mantissa) length presents a number of problems with the stability of the calculations in question.

APPENDIX IX

SHELL62

```
$CODE LIST DBUGA SUMRY SAVE ZIP
BEGIN
%        THIS IS 62 ZONE (OR LESS) WIDE SHELL IN GLYPNIR.
CINT IMAX,JMAX,JBLK,J,J1,J2;
CREAL PROB,CYCLE,T,DT,DTLAST,WCA,WCB,WCC,WCD,WCF,PI,FTH,STABLFCTR;
CU REAL RELERRBEFORE,CYCLASTPRINT,MAXRELERR,GO,TOZONE;
PREAL DX,TAU,CS,WPA,WPB,WPC,RC,DTH;
PREAL VECTOR DY[0];
BOOLEAN BOTTOMREFLECTIVE, RIGHTREFLECTIVE, TOPREFLECTIVE;
BOOLEAN REZONETOP,REZONERIGHT,REZONEBOTTOM,IMODE;
LABEL SHELL.AGAIN,STOP,FIN;
PREAL STDT;% THIS CARD IN OUTER BLK FOR STD TIME CDT.
% THE FOLLOWING CARD DEFINES THE COMMON BLOCK.
PREAL Z; PREAL VECTOR U,V,AMX,AIX,P[7]; PREAL X; PREAL VECTOR Y[0];
%
FILE I4DISKO="SHELL"/"DTG2" ( 43 ROWS FULL);
FILE I4DISKI="SHELL"/"DTG3" ( 43 ROWS FULL);
FILE LINE (1 ROW);
%
%
%        FOLLOWING ARE ALL SUBROUTINES.
%
%
SUBROUTINE PUTCPR(CREAL X,PREAL OUT Z,CINT I);  BEGIN
     IF PEN=I THEN Z←X;
END;
%
%
%
%
SUBROUTINE PUTCPI(CINT X,PREAL OUT Z,CINT I);  BEGIN
     CREAL A; A←X; IF PEN=I THEN Z←A;
END;
%
%
%
%
PREAL SUBROUTINE LN AS RGA (PREAL X AS RGA);  BEGIN
$ INCINS=100, ASKRATCH=18
$STAPE2=GLYPNIR/GLN SERIAL;
END;
%
%
%
%
PREAL SUBROUTINE EXP AS RGA (PREAL X AS RGA);  BEGIN
$ INCINS=100, ASKRATCH=17
$STAPE2=GLYPNIR/GEXP SERIAL;
END;
%
%
%
%
PREAL SUBROUTINE SQRT AS RGA (PREAL X AS RGA);  BEGIN
$ INCINS=110, ASKRATCH=14
$STAPE2=GLYPNIR/GSQRT SERIAL;
END;
%
%
%
```

```
%
PREAL SUBROUTINE ALOG10(PREAL X); BEGIN
      ALOG10←0.4342944819032518×LN(X);
END;
%
%
%
%
PREAL SUBROUTINE POWEROF10 (PREAL N); BEGIN
      %   POWEROF10=10.**N.
      POWEROF10←EXP(2.302585092994046×N);
END;
%
%
%
%
SUBROUTINE POWER(PREAL X,PREAL OUT XX,PINT OUT N); BEGIN
      XX←ALOG10(ABS(X));
      N←(XX-0.5×SIGN(XX)); %DONT LET GLYPNIR ROUND.
      IF(N<0) THEN N←N-1;
      XX←X×POWEROF10(-N);
END;
%
%
%
%
SUBROUTINE NEGATIVEMASS(CINT J); BEGIN
      CINT J.I;
       MODE←TMODE;
      LOOP J.I←0,1,JMAX DO BEGIN
            IF(AMX[JJ]≤0.)THEN BEGIN
                  SIMWRITE(LINE[PAGE],"NEG MASS","J =",J);
                  SIMWRITE(LINE,"AMX[*]",AMX[*]);
                  ERROR(0);
            END;
      END;
END; % NEGATIVEMASS.
%
%
%
%
SUBROUTINE NEGATIVEENERGY(CINT J); BEGIN
      CINT J.I;
       MODE←TMODE;
      LOOP J.I←0,1,JMAX DO BEGIN
            IF(AIX[JJ]≤0.)THEN BEGIN
                  SIMWRITE(LINE[PAGE],"NEG ENER","J =",J);
                  SIMWRITE(LINE,"AIX[*]",AIX[*]);
                  ERROR(0);
            END;
      END;
      MODE←TRUE;
      SIMWRITE(IADISK1,7);
      SIMWRITE(LINE,"MASS AND ENERGY OK AT J = ",J);
END; % NEGATIVEENERGY.
%
%
%
%
CREAL SUBROUTINE SUMROW(PREAL X); BEGIN
      CINT J,TWOTOTHEJ;BOOLEAN OLDMODE;
```

```
        OLDMODE←MODE;
        MODE←NOT MODE;
        X←0.;
        MODE←TRUE;
        TWOTOTHEJ←1.;
        LOOP J←0,1,5 DO  BEGIN
             X←X+RTR(TWOTOTHEJ,MODE,X);
             TWOTOTHEJ←SHIFTL(1,TWOTOTHEJ);
        END;
        SUMROW←X;
        MODE←OLDMODE;
   END; % SUMROW
   %
   %
   %
   %
   SUBROUTINE ERROR(CINT J);  BEGIN
        SIMWRITE(LINE[PAGE],"ERROR =",J);
        WCA←0.;
        WCA←1.0+1.0/WCA; %ABORT.
   END; %ERROR.
   %
   %
   %
   %
   SUBROUTINE CDT;  BEGIN
        PREAL MINDIMENSION;
        CREAL BFR,TNEXT,WDA,WDB,WDY; CINT LP,J; BOOLEAN M1,M2;
        WDA←1.0@-20;
        LOOP J←0,1,JMAX DO  BEGIN
             MODE←TRUE;
             J1←J DIV 64;
             J2←J-64×J1;
             WDY←GRABONE(DY[J1],J2);
             MODE←TMODE;
             CS←AMX[J]/(WDY×TAU);
             ES(J);%ES RETURNS UPDATED P AND CS FOR THE ROW J.
                  MINDIMENSION←DX;
             IF(WDY<DX) THEN MINDIMENSION←WDY;
             WDB← MAX(CS/MINDIMENSION); % TIME FOR SOUND TO CROSS MIN CELL
             IF (WDA<WDB) THEN WDA←WDB;
             WDB← MAX(ABS(U[J])/DX);% TIME FOR MATERIAL TO CROSS CELL IN X
                                     % DIRECTION
             IF (WDA<WDB) THEN WDA←WDB;
             WDB← MAX(ABS(V[J])/WDY); % TIME FOR MATERIAL TO CROSS CELL IN Z
                                         % DIRECTION
             IF (WDA<WDB) THEN WDA←WDB;
        END;
        MODE←TRUE;
        DT←0.5×STABLFCTR/WDA;
        DT←1.0@-8;
        T←T+DTLAST;
        POWER(T,BFR,LP);
        TNEXT←(T+DT)×POWEROF10(-LP);
        M1← (TNEXT>STDT) AND BOOLEAN(17777777777776000000000(8));
        M2←SHIFTL(1,M1);
        M1←(M1 OR M2) AND NOT (M1 AND M2);
        IF(M1 AND (BFR<STDT)) THEN DT←(STDT-BFR)×POWEROF10(LP);
        DTLAST←DT;
        CYCLE←CYCLE+1;
   END; %CDT.
```

```
%
%
%
%
%
%
%
%
SUBROUTINE REZONE (CINT J);  BEGIN
     ERROR(100);
END;
%
%
%
%
SUBROUTINE ES(CINT K);
BEGIN
% ES USES RHO [IN CS ARRAY], AND AIX[K].
% IT RETURNS UPDATED P[K] AND CS.
% MODIFICATION OF AIR TO EXTEND TO 2.E13 ERGS/GM.  11 MAY 69
     PREAL E,LNEROVROZ,FO,FON,FN,WS,BETA,FE,GMONE;
     PREAL RHO AS CS;
     E←1.0@-10×AIX[K];
     LNEROVROZ←LN(773.39520495×RHO);
     WS←1.0/(1.0+EXP((-8.5-0.15504314×LNEROVROZ+E)×EXP(-0.05×LNEROVROZ
          +0.02531780798))));
     FO ←WS ×EXP(-0.2242152a664×E));
     FON←(1.0-WS )×EXP(-0.1508295625 9×E));
     BETA←0.;
     IF(E>1.)THEN BETA← (0.0069487×WS+0.0138974)×LN(E);
     WS←1.0/(1.0+EXP((-E+EXP(0.0157×LNEROVROZ+3.80666248 9))×EXP(-0.085
          ×LNEROVROZ-1.38629436))));
     FN←WS×EXP(-0.039215686275×E));
     BETA←BETA×(1.0-WS)+0.045×WS;
     WPA←30.0+3.474356×LNEROVROZ;
     IF WPA <6.0 THEN WPA←6.0;
     WPA←(160.0-E)/WPA;
     FE←1.0/(1.0+EXP(WPA));
     GMONE←(0.161+0.255×FO+0.280×FON+0.137×FN+0.05×FE)×EXP(BETA
          ×LNEROVROZ);
     P[K]←GMONE×RHO×AIX[K];
     CS←SQRT((1.0+GMONE)×P[K]/RHO);
END; %ES.
%
%
%
%
SUBROUTINE INPUT;  BEGIN
     MODE←TRUE;
     SIMREAD(I4DISKO,Z);
     PROB  ←GRABONE(Z,0);
     CYCLE←GRABONE(Z,1);
     CYCLASTPRINT←CYCLE;
     T     ←GRABONE(Z,2);
     DT    ←GRABONE(Z,3);
     DTLAST←GRABONE(Z,4);
     T←T-DTLAST;
     CYCLE←CYCLE-1;
     IMAX ←GRABONE(Z,5);
     JMAX ←GRABONE(Z,6);
     ETH←GRABONE(Z,7);
```

```
        STABLFCTR←GRABONE(Z,8);
        IF(STABLFCTR≤0.) THEN STABLFCTR←1.;
        GO←GRABONE(Z,9);
        GO←0.0;
        TOZONE←GRABONE(Z,10);
        JBLK←JMAX DIV 64;
        TOPREFLECTIVE ← FALSE;
        RIGHTREFLECTIVE ← FALSE;
        BOTTOMREFLECTIVE ← FALSE;
        IF(GRABONE(Z,11)≠0.) THEN TOPREFLECTIVE←TRUE;
        IF(GRABONE(Z,12)≠0.) THEN RIGHTREFLECTIVE←TRUE;
        IF(GRABONE(Z,13)≠0.) THEN BOTTOMREFLECTIVE←TRUE;
        RELERRBEFORE←GRABONE(Z,14);
        MAXRELERR←GRABONE(Z,15);
        DX←X−RTR(1,MODE,X);
        RC←0.5×(X+RTR(1,MODE,X));
        MODE←BOOLEAN(−0);
        MODE←SHIFTR(IMAX+1,MODE);
        DX←RTR(2,,DX);
        RC←RTR(1,,X)+0.5×DX;
        MODE←BOOLEAN(−0);
        DX← RTL(1,,DX);
        RC←−RTL(1,,RC);
        MODE←TRUE;
        IMODE←NOT BOOLEAN (−0) AND PENSIMAX;
        WPA←X×X;
        TAU←  _PI×(WPA−RTR(1,MODE,WPA));
        LOOP J←0,1,JBLK DO BEGIN
              DY[J]←RTL(1,MODE ,Y[J]);
              IF(J≠JBLK) THEN BEGIN
                    IF BOOLEAN(1) THEN DY[J]← RTL(1,,Y[J+1]);
              END;
              DY[J]←DY[J]−Y[J];
        END;
  END; %INPUT.
  %
  %
  %
  %
  SUBROUTINE DATA;  BEGIN
        CODE
        USE       STDT,PI;  BEGIN
              HEREIAM; SET ;
              ORG  PI;
              DATA 3.14159265358979З;
              ORG  STDT;
              DATA 1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.3,2.5,2.6,
                   2.8,3.0,3.2,3.4,3.6,3.8,4.0,4.2,4.5,4.8,5.0,5.3,5.6,6.0,
                   6.5,7.0,7.5,8.0,8.5,9.0,9.5,10.0;
              ORG  HEREIAM;
        END CODE;
  END; %DATA.
  %
  %
  %
  %
  SUBROUTINE PHI;  BEGIN
        CINT PASS; CREAL G;
        PREAL VA,VR,PA,PR,PB,URR;
        MODE←TRUE;
        DTH←0.;
```

```
URR←0.0;
IF RIGHTREFLECTIVE THEN
ERROR(1)
ELSE
BEGIN
      % SET P AT AXIS AND RIGHT
      MODE←BOOLEAN(-0);
      LOOP J←0,1,JMAX DO  P[J]←RTL(1,,P[J]);
      MODE←SHIFTR(IMAX+1,MODE);
      LOOP J←0,1,JMAX DO P[J]←RTR(2,,P[J]);
END;
LOOP PASS←1,1,2 DO  BEGIN
      MODE←TRUE;
      IF BOTTOMREFLECTIVE THEN  BEGIN
            VB←0,;
            PB←P[0];
      END
      ELSE BEGIN
            VB←V[0];
            WCA← GRABONE(DY[0],0);
            WCB← GRABONE(DY[0],1);
            WCC←1./(WCA+WCB);
            PB←(P[1]×WCA+P[0]×WCB)×WCC;
            DTH←DTH+0.5×TAU×WCA×V[0]×DT×(P[0]+P[1])×WCC;
      END;
      IF RIGHTREFLECTIVE THEN ERROR(2)
      ELSE BEGIN
            % SET U AT AXIS AND RIGHT.
            MODE←BOOLEAN(-0);
            LOOP J←0,1,JMAX DO U[J]←-RTL(1,,U[J]);
            MODE←SHIFTR(IMAX+1,MODE);
            WCA←RTR(1,,RC)/RC;
            LOOP J←0,1,JMAX DO U[J]←WCA×RTR(1,,U[J]);
      END;
      MODE←¬MODE OR BOOLEAN(-0);
      WPA←1.0/(DX+RTL(1,,DX));
      WPB←RC×RTL(1,,DX)×WPA;
      WPC←RTL(1,,RC)×DX×WPA;
      MODE←TRUE;
      WCA←GRABONE(DY[0],0);
      MODE←¬MODE;
      LOOP J←0,1,JMAX-1 DO  BEGIN
            IF(AMX[J]≤0.) THEN ERROR(3);
            URR←U[J]×WPB+RTL(1,,U[J])×WPC;
            MODE←TRUE;
            J1←(J+1) DIV 64;
            J2←(J+1)-64×J1;
            WCB←GRABONE(DY[J1],J2);
            MODE←IMODE;
            WCC←1.0/(WCA+WCB);
            WCD←WCA×WCC;
            WCE←WCB×WCC;
            VA←V[J]×WCE+V[J+1]×WCD;
            AIX[J]←AIX[J]+P[J]×DT×(0.5×(VB-VA)×TAU+(RTR(1,,URR)-URR)
                  ×PI×WCA)/AMX[J];
            IF(PASS=1) THEN  BEGIN
                  PA←P[J]×WCE+P[J+1]×WCD;
                  MODE←TRUE;
                  G←6,356708;
                  G←G/(G+GRABONE(Y[J1],J2)-0.5×WCA);
                  G←G0×G×G;
```

```
                    MODE←IMODE OR BOOLEAN(-0);
                    PR←(P[J]×RTL(1,,DX)+RTL(1,,P[J])×DX)×WPA;
                    MODE←IMODE;
                    U[J]←U[J]+TAU×WCA×(RTR(1,,PR)-PR)×DT/(DX×AMX[J]);
                    V[J]←V[J]+TAU×WCA×(PB-PA)×DT/(WCA×AMX[J])-G×DT;
                    PB←PA;
               END;
               VB←VA;
               WCA←WCB;
          END; % J LOOP
          J←JMAX;
          IF(AMX[J]≤0.) THEN ERROR(4);
          URR←U[J]×WPB+RTL(1,,U[J])×WPC;
          IF TOPREFLECTIVE THEN ERROR(5)
          ELSE BEGIN
               VA←V[J];
               % PA←PB; % ALREADY DONE ABOVE
               DTH←DTH-0.5×TAU×WCB×V[J]×DT×(P[J]+P[J-1])×WCC;
          END;
          AIX[J]←AIX[J]+P[J]×DT×(0.5×(VB-VA)×TAU+(RTR(1,,URR)-URR)
               ×PI×WCA)/AMX[J];
          IF PASS=1 THEN BEGIN
               MODE←MODE OR BOOLEAN(-0);
               PR←(P[J]×RTL(1,,DX)+RTL(1,,P[J])×DX)×WPA;
               MODE←IMODE;
               U[J]←U[J]+TAU×WCA×(RTR(1,,PR)-PR)×DT/(DX×AMX[J]);
          END;
          % DONE WITH WHOLE GRID
     END; % PASS LOOP.
     ETH←ETH+SUMROW(DTH);
END; % PH1
SUBROUTINE PH2;  BEGIN
     REAL SPECIFICENERGY,DONORTOPU,DONORTOPV,DONORTOPSPECIFICENERGY,
     VATTOPINTERFACE,DONORMASSDIVIDEDBYDY,MASSTHRUTOP,MASSTHRUBOTTOM,
     DONORRIGHTU,DONORRIGHTV,DONORRIGHTSPECIFICENERGY,
     DONORMASSDIVIDEDBYDX,UATRIGHTINTERFACE,MASSTHRURIGHT,AMXNEW,
     ONEOVERAMXNEW,UMOMENTUMTHRUTOP,VMOMENTUMTHRUTOP,UMOMENTUMTHRURIGHT,
     VMOMENTUMTHRURIGHT,ENERGYTHRURIGHT,ENERGYTHRUTOP,ENERGYTHRUBOTTOM,
     UMOMENTUMTHRUBOTTOM,VMOMENTUMTHRUBOTTOM;
     REZONETOP←FALSE;REZONERIGHT←FALSE;REZONEBOTTOM←FALSE;
     DTH←0.;
      MODE←TRUE;
     WCA←GRABONE(DY[0],0);
     % SET AXIS BOUNDARY CONDITIONS
     MODE←BOOLEAN(-0);
     MASSTHRURIGHT←0.;
     % SET BOTTOM BOUNDARY CONDITIONS
     MODE←TRUE;
     IF BOTTOMREFLECTIVE THEN  BEGIN
          MASSTHRUBOTTOM←0.;
          ENERGYTHRUBOTTOM←0.;
          UMOMENTUMTHRUBOTTOM←0.;
          VMOMENTUMTHRUBOTTOM←0.;
     END
     ELSE BEGIN
          MASSTHRUBOTTOM←V[0]×AMX[0]×DT/WCA;
          IF(MASSTHRUBOTTOM>0.) THEN MASSTHRUBOTTOM←0.;
          IF(-MASSTHRUBOTTOM>TOZONE×AMX[0]) THEN REZONEBOTTOM←TRUE;
          ENERGYTHRUBOTTOM←MASSTHRUBOTTOM×(AIX[0]+0.5×(U[0]×U[0]+V[0]×
               V[0]));
          DTH←ENERGYTHRUBOTTOM;
```

```
      UMOMENTUMTHRUBOTTOM←MASSTHRUBOTTOM×U[0];
      VMOMENTUMTHRUBOTTOM←MASSTHRUBOTTOM×V[0];
END;
IF RIGHTREFLECTIVE THEN ERROR(6)
ELSE BEGIN
      % SET RIGHT BOUNDARY CONDITIONS
      MODE←SHIFTR(IMAX+1,MODE);
      LOOP J←0,1,JMAX DO    BEGIN
            U[J]←RTR(1,,U[J]);
            IF(U[J]<0.) THEN U[J]←-U[J];
      END;
END;
 MODE←IMODE;
LOOP J←0,1,JMAX DO  BEGIN
      % COMPUTE QUANTITIES AT TOP INTERFACE
      SPECIFICENERGY←AIX[J]+0.5×(U[J]×U[J]+V[J]×V[J]);
      DONORTOPU←U[J];
      DONORTOPV←V[J];
      DONORTOPSPECIFICENERGY←SPECIFICENERGY;
      DONORMASSDIVIDEDBYDY←AMX[J]/WCA;
      IF J<JMAX THEN  BEGIN
            MODE←TRUE;
            J1←(J+1) DIV 64;
            J2←(J+1)-64×J1;
            WCB←GRABONE(DY[J1],J2);
            MODE←IMODE;
            VATTOPINTERFACE←0.5×(V[J]+V[J+1]);
            VATTOPINTERFACE←VATTOPINTERFACE/(1.0+(V[J+1]-V[J])/WCA
                  ×DT);
            IF (VATTOPINTERFACE<0.) THEN  BEGIN
                  DONORMASSDIVIDEDBYDY←AMX[J+1]/WCB;
                  DONORTOPU←U[J+1];
                  DONORTOPV←V[J+1];
                  DONORTOPSPECIFICENERGY←AIX[J+1]+0.5×(U[J+1]×U[J+1]+
                        V[J+1]×V[J+1]);
            END;
            MASSTHRUTOP←VATTOPINTERFACE×DT×DONORMASSDIVIDEDBYDY;
      END
      ELSE BEGIN     % DO TOP ROW DIFFERENTLY.
            IF TOPREFLECTIVE THEN ERROR(7)
            ELSE BEGIN
                  VATTOPINTERFACE←V[J];
                  MASSTHRUTOP←VATTOPINTERFACE×DT×DONORMASSDIVIDEDBYDY;
                  IF(MASSTHRUTOP<0.) THEN MASSTHRUTOP←0.;
                  IF(MASSTHRUTOP>TOZONE×AMX[J]) THEN REZONETOP←TRUE;
                  DTH←DTH+MASSTHRUTOP×DONORTOPSPECIFICENERGY;
            END;
      END;
      % COMPUTE QUANTITIES AT RIGHT INTERFACE
      DONORRIGHTU←U[J];
      DONORRIGHTV←V[J];
      DONORRIGHTSPECIFICENERGY←SPECIFICENERGY;
      DONORMASSDIVIDEDBYDX←AMX[J]/(DX×RC);
      UATRIGHTINTERFACE←0.5×(U[J]+RTL(1,,U[J]));
      UATRIGHTINTERFACE←UATRIGHTINTERFACE/(1.0+(RTL(1,,U[J])-U[J])
            /DX×DT);
      IF (UATRIGHTINTERFACE<0.) THEN  BEGIN
            DONORMASSDIVIDEDBYDX←RTL(1,,DONORMASSDIVIDEDBYDX);
            DONORRIGHTU←RTL(1,,DONORRIGHTU);
            DONORRIGHTV←RTL(1,,DONORRIGHTV);
            DONORRIGHTSPECIFICENERGY←
```

```
                    RTL(1,,DONORRIGHTSPECIFICENERGY));
            END;
            MASSTHRURIGHT←UATRIGHTINTERFACE×DT×DONORMASSDIVIDEDBYDX×X;
            MODE←TRUE;
            IF(GRABONE(MASSTHRURIGHT,IMAX)>TOZONE×GRABONE(AMX[J],IMAX))
            THEN REZONERIGHT←TRUE;
            MODE←IMODE;
            AMXNEW←AMX[J]-MASSTHRUTOP+MASSTHRUBOTTOM-MASSTHRURIGHT+
                 RTR(1,,MASSTHRURIGHT));
            ONEOVERAMXNEW←1.0/AMXNEW;
            % MOMENTUM FLUXES AT TOP AND RIGHT
            UMOMENTUMTHRUTOP←MASSTHRUTOP×DONORTOPU;
            UMOMENTUMTHRURIGHT←MASSTHRURIGHT×DONORRIGHTU;
            VMOMENTUMTHRUTOP←MASSTHRUTOP×DONORTOPV;
            VMOMENTUMTHRURIGHT←MASSTHRURIGHT×DONORRIGHTV;
            ENERGYTHRUTOP←MASSTHRUTOP×DONORTOPSPECIFICENERGY;
            ENERGYTHRURIGHT←MASSTHRURIGHT×DONORRIGHTSPECIFICENERGY;
            IF PEN=IMAX THEN DTH←DTH-ENERGYTHRURIGHT;
            % UPDATE U
            U[J]←(AMX[J]×U[J]-UMOMENTUMTHRUTOP+UMOMENTUMTHRUBOTTOM-
            UMOMENTUMTHRURIGHT+RTR(1,,UMOMENTUMTHRURIGHT))×ONEOVERAMXNEW;
            % UPDATE V
            V[J]←(AMX[J]×V[J]-VMOMENTUMTHRUTOP+VMOMENTUMTHRUBOTTOM-
            VMOMENTUMTHRURIGHT+RTR(1,,VMOMENTUMTHRURIGHT))×ONEOVERAMXNEW;
            % UPDATE AIX
            AIX[J]←(AMX[J]×SPECIFICENERGY-ENERGYTHRUTOP+ENERGYTHRUBOTTOM-
            ENERGYTHRURIGHT+RTR(1,,ENERGYTHRURIGHT)-0.5×AMXNEW×(U[J]×U[J]+
                 V[J]×V[J]))×ONEOVERAMXNEW;
            % UPDATE AMX
            AMX[J]←AMXNEW;
            % STORE BOTTOM QUANTITIES FOR NEXT ROW
            UMOMENTUMTHRUBOTTOM←UMOMENTUMTHRUTOP;
            VMOMENTUMTHRUBOTTOM←VMOMENTUMTHRUTOP;
            ENERGYTHRUBOTTOM←ENERGYTHRUTOP;
            MASSTHRUBOTTOM←MASSTHRUTOP;
        END; % J LOOP
        ETH←ETH+SUMROW(DTH));
END; % PH2
%
%
%
%
SUBROUTINE OUTPUT;  BEGIN
        CU REAL TOTALENERGYNOW,RELERRNOW,ENGYCHECK,NUMBEROFCYCLES;
        MODE←IMODE;
        WPA←0.0;
        LOOP J←0,1,JMAX DO WPA←WPA+AMX[J]×(AIX[J]+0.5×(U[J]×U[J]+V[J]×V[J])
             );
        TOTALENERGYNOW←SUMROW(WPA));
        RELERRNOW←(TOTALENERGYNOW-ETH)/ETH;
        NUMBEROFCYCLES←CYCLE-CYCLASTPRINT;
        IF (NUMBEROFCYCLES=0.0) THEN NUMBEROFCYCLES←1.0;
        ENGYCHECK←(RELERRNOW-RELERRBEFORE)/NUMBEROFCYCLES;
        RELERRBEFORE←RELERRNOW;
        CYCLASTPRINT←CYCLE;
        MODE←TRUE;
        PUTCPR(CYCLE,Z,1));
        PUTCPR(T     ,Z,2));
        PUTCPR(DT    ,Z,3));
        PUTCPR(DTLAST,Z,4));
        PUTCPR(ETH,Z,7));
```

```
      PUTCPR(STARLFCTR,7,8);
      PUTCPR(GO,7,9);
      PUTCPR(TOZONF,Z,10);
      PUTCPR(0,7,11);PUTCPR(0,Z,12);PUTCPR(0,Z,13);
      IF TOPREFLECTIVE THEN PUTCPR(1,Z,11);
      IF RIGHTREFLECTIVE THEN PUTCPR(1,Z,12);
      IF BOTTOMREFLECTIVE THEN PUTCPR(1,Z,13);
      PUTCPR(RELERRBEFORE,Z,14);
      SIMWRITE(TADISK1,Z);
      %IF(RELERRNOW>MAXRELERR) THEN ERROR(200);
END; %OUTPUT.
%
%
%
%
%        MAIN PROGRAM STARTS HERE.
%
%
SHELL: OPEN(TADISK0,0,1);
        OPEN(TADISK1,0,1);
        INPUT;
        AGAIN:  CDT;
        OUTPUT;
        PH1;
        NEGATIVEMASS(3);NEGATIVEENERGY(3);
        PH2;
        NEGATIVEMASS(4);NEGATIVEENERGY(4);
        % AT THIS POINT SHOULD GO
        % IF(REZONETOP) THEN REZONE(1);
        % IF(REZONERIGHT) THEN REZONE(2);
        % IF(REZONEBOTTOM) THEN REZONE(3);
        GO TO AGAIN;
        %
        STOP;FIN;END.
```

APPENDIX X

SHELLN

```
$CODE LIST DBUGA SUMRY SAVE
BEGIN
$ THIS IS IMAXxJMAX ZONE SHELL.
BOOLEAN MODE1;
CINT IMAX,JMAX,JBLK,J,J1,J2,IBLK,IBLKP1,I,J;
CREAL PROB,CYCLE,T,DT,DTLAST,WCA,WCB,WCC,WCD,WCE,PI,ETH,STABLFCTR;
CU REAL RELERRBEFORE,CYCLASTPRINT,MAXRELERR,GO,TOZONE;
PREAL         CS,WPA,WPB,WPC,    DTH;
PREAL VECTOR DY[0],DX,RC,TAU[2];
BOOLEAN BOTTOMREFLECTIVE, RIGHTREFLECTIVE, TOPREFLECTIVE;
BOOLEAN REZONETOP,REZONERIGHT,REZONEBOTTOM;
LABEL SHELL,AGAIN,STOP,FIN;
PREAL STDT;% THIS CARD IN OUTER BLK FOR STD TIME CUT.
%        THE FOLLOWING CARDS DEFINE THE COMMON.
PREAL Z;
PREAL VECTOR UDMY[0],U[20],VDMY[0],V[20],AMXDMY[0],AMX[20],
         AIXDMY[0],AIX[20],PDMY[0],P[20];
PREAL VECTOR X[2],Y[0];
%
FILE I4DISK0="SHELL"/"BTOI" (108 ROWS FULL);
FILE I4DISK1="SHELL"/"ITOB" (108 ROWS FULL);
FILE LINE (1 ROW);
%
%
%        FOLLOWING ARE ALL SUBROUTINES.
%
%
SUBROUTINE PUTCPR(CREAL X,PREAL OUT Z,CINT I);
BEGIN IF PEN=I THEN Z←X; END;
%
%
%
%
SUBROUTINE PUTCPI(CINT X,PREAL OUT Z,CINT I);
BEGIN CREAL A; A←X; IF PEN=I THEN Z←A; END;
%
%
%
%
PREAL SUBROUTINE LN AS RGA (PREAL X AS RGA);
BEGIN
$ INCINS=100, ASKRATCH=18
$$TAPE2=GLYPNIR/GLN SERIAL;
END;
%
%
%
%
PREAL SUBROUTINE EXP AS RGA (PREAL X AS RGA);
BEGIN
$ INCINS=100, ASKRATCH=17
$$TAPE2=GLYPNIR/GEXP SERIAL;
END;
%
%
%
%
PREAL SUBROUTINE SQRT AS RGA (PREAL X AS RGA);
BEGIN
$ INCINS=110, ASKRATCH=14
$$TAPE2=GLYPNIR/GSQRT SERIAL;
```

```
END;
%
%
%
%
PREAL SUBROUTINE ALOG10(PREAL X);
BEGIN
ALOG10←0.4342944819032518×LN(X);
END;
%
%
%
%
PREAL SUBROUTINE POWEROF10 (PREAL N);
BEGIN
%          POWEROF10=10.**N.
POWEROF10←EXP(2.302585092994046×N);
END;
%
%
%
%
SUBROUTINE POWER(PREAL X,PREAL OUT XX,PINT OUT N);
BEGIN
XX←ALOG10(ABS(X));
N←(XX-0.5×SIGN(XX)); %DONT LET GLYPNIR ROUND.
IF(N<0) THEN N←N-1;
XX←X×POWEROF10(-N);
END;
%
%
%
%
SUBROUTINE ERROR(CINT J);
BEGIN
  SIMWRITE(LINE[PAGE],"ERROR =",J);
  WCA←0.;
  WCA←1.0+1.0/WCA; %ABORT.
END; %ERROR.
%
%
%
%
SUBROUTINE NEGATIVEMASS(CINT J);
BEGIN
CINT II,JJ;
LOOP JJ←0,1,JMAX DO BEGIN
LOOP II←0,1,IBLK DO BEGIN
K←JJ×IBLKP1+II;
IF(AMX[K]≤0.) THEN BEGIN
SIMWRITE(LINE[PAGE],"NEG MASS","J =",J);
SIMWRITE(LINE,"AMX[*]",AMX[*]);
ERROR(0);
END;END;END;
END; % NEGATIVEMASS.
%
%
%
%
SUBROUTINE NEGATIVEENERGY(CINT J);
BEGIN
```

```
CINT II,JJ;
LOOP JJ←0,1,JMAX DO BEGIN
LOOP II←0,1,IBLK DO BEGIN
K←JJ×IBLKP1+II;
IF(AIX[K]≤0.) THEN BEGIN
SIMWRITE(LINE[PAGE],"NEG ENER","J =",J);
SIMWRITE(LINE,"AIX[*]",AIX[*]);
ERROR(0);
END;END;END;
SIMWRITE(LINE,"MASS AND ENERGY OK AT J = ",J);
END; % NEGATIVEENERGY.
%
%
%
%
CREAL SUBROUTINE SUMROW(PREAL X);
BEGIN
      CINT J,TWOTOTHEJ;BOOLEAN OLDMODE;
      OLDMODE←MODE;
      MODE←NOT MODE;
      X←0.;
      MODE←TRUE;
      TWOTOTHEJ←1.;
      LOOP J←0,1,5 DO
      BEGIN
            X←X+RTR(TWOTOTHEJ,MODE,X);
            TWOTOTHEJ←SHIFTL(1,TWOTOTHEJ);
      END;
      SUMROW←X;
      MODE←OLDMODE;
END; % SUMROW
%
%
%
%
CREAL SUBROUTINE CMIN(CREAL X, CREAL Y);
BEGIN
CMIN←X;
IF Y<X THEN CMIN←Y;
END;
%
%
%
%
SUBROUTINE INPUT;
BEGIN
SIMREAD(I4DISKO,Z);
PROB ←GRABONE(Z,0);
CYCLE←GRABONE(Z,1);
CYCLASTPRINT←CYCLE;
T    ←GRABONE(Z,2);
DT   ←GRABONE(Z,3);
DTLAST←GRABONE(Z,4);
IMAX ←GRABONE(Z,5);
JMAX ←GRABONE(Z,6)-1;
ETH←GRABONE(Z,7);
STABLFCTR←GRABONE(Z,8);
IF(STABLFCTR≤0.) THEN STABLFCTR←1.;
G0←GRABONE(Z,9);
IF(G0=0.) THEN G0←980.;
TUZONE←GRABONE(Z,10);
```

```
IBLK←(IMAX+1) DIV 64;
IBLKP1←IBLK+1;
JBLK←JMAX DIV 64;
      TOPREFLECTIVE ← FALSE;
      RIGHTREFLECTIVE ← FALSE;
      BOTTOMREFLECTIVE ← FALSE;
IF(GRABONE(Z,11)≠0.) THEN TOPREFLECTIVE←TRUE;
IF(GRABONE(Z,12)≠0.) THEN RIGHTREFLECTIVE←TRUE;
IF(GRABONE(Z,13)≠0.) THEN BOTTOMREFLECTIVE←TRUE;
RELERRBEFORE←GRABONE(Z,14);
MAXRELERR←GRABONE(Z,15);
T←T-DTLAST;
CYCLE←CYCLE-1;
LOOP I←0,1,IBLK DO
BEGIN
WRA←RTR(1,MODE,X[I]);
IF(I≠0) THEN IF BOOLEAN(-0) WRA←RTR(1,,X[I-1]);
RC[I]←0.5×(X[I]+WRA);
DX[I]←X[I]-WRA;
TAU[I]←PI×(X[I]×X[I] -WRA×WRA);
END;
MODE←BOOLEAN(-0);
DX[0]←-RTL(1,,DX[0]);
RC[0]←-RTL(1,,RC[0]);
LOOP J←0,1,JBLK DO
BEGIN
     DY[J]←RTL(1,MODE ,Y[J]);
     IF(J≠JBLK) THEN
     BEGIN
          IF BOOLEAN(1) THEN DY[J]← RTL(1,,Y[J+1]);
     END;
     DY[J]←DY[J]-Y[J];
END;
END; %INPUT,
%
%
%
%
SUBROUTINE ES(CINT I,CINT J);
% MODIFICATION OF AIR TO EXTEND TO 2.E13 ERGS/GM.   11 MAY 69
BEGIN
PREAL E,RHO,ROVROZ,LNEROVROZ,POWER,FO,FON,FN,WS,E1,E2,BETA,FE,GMONE;
BOOLEAN OLDMODE;
K←J×(IBLK+1)+I;
E←1.0@-10×A[X[K]];
J1←J DIV 64;
J2←J-64×J1;
OLDMODE←MODE;
MODE←TRUE;
WCA←TAU×GRABONE(DY[J1],J2);
MODE←OLDMODE;
RHO←AM×[J]/WCA;
ROVROZ←773.3952×RHO;
LNEROVROZ←LN(ROVROZ);
POWER←0.4342944819032S2×LNEROVROZ;%  POWER←ALOG10(ROVROZ);
E1 ←1.025641×(8.5-E);
     WS ←1.0/(1.0+EXP(-(8.5-0.357×POWER-E)/(0.975×EXP(0.05×LNEROVROZ))));
     FO ←WS ×EXP(-0.2242152×E);
     FON←(1.0-WS )×EXP(-0.1506295×E);
BETA←0.;
IF(E>1.)THEN BETA← (0.048×WS +0.032×(1.-WS))×ALOG10(E);
```

```
E2 ←0.3333333333333333×(E−40.);
    WS←1.0/(1.0+EXP(−0.25×(E−45.0×EXP(0.0157×LNERDVRDZ))×
          EXP(−0.085×LNERDVRDZ)));
    FN←WS×EXP(−0.0392156×E);
WPA←30.0−6.0×POWER;
IF WPA >6.0 THEN WPA←6.0;
WPA←(E−160.0)/WPA;
BETA←BETA×(1.0−WS)+0.045×WS;
FE←0.;
IF WPA≥−5.0 THEN FE←1.0/(1.0+EXP(−WPA));
GMONE←(0.161+0.255×FD +0.280×FDN+0.137×FN+0.050×FE)×EXP(BETA×LNERDVRDZ);
P[K]←GMONE×RHO×AIX[K];
CS←SQRT((1.0+GMONE)×P[K]/RHO);
END; %ES.
%
%
%
%
SUBROUTINE DATA;
BEGIN
          CODE
USE       STDT,PI;
          BEGIN
HEREIAM:  SET ;
          ORG  PI;
          DATA 3.141592653589793;
          ORG  STDT;
       DATA 1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.3,2.5,2.6,2.8,
          3.0,3.2,3.4,3.6,3.8,4.0,4.2,4.5,4.8,5.0,5.3,5.6,6.0,6.5,7.0,
          7.5,8.0,8.5,9.0,9.5,10.0;
          ORG  HEREIAM;
          END CODE;
END; %DATA.
%
%
%
%
SUBROUTINE CDT;
BEGIN
FREAL MINDIMENSION;
CREAL BER,TNEXT,WDA,WDB,WDY; CINT LP,J; BOOLEAN M1,M2;
WDA←1.0@20;
LOOP J←0,1,JMAX DO
    BEGIN
     J1←J DIV 64;
     J2←J−64×J1;
     WDY←GRABONE(DY[J1],J2);
LOOP I←0,1,IBLK DO
BEGIN
    K←J×IBLK+I;
MODE1←PEN+64×ISIMAX;
IF(IBLK=0) THEN MODE1←MODE1 AND NOT BOOLEAN(−0);
MODE←MODE1;
    ES(I,J);%ES RETURNS UPDATED P AND CS FOR THE ROW K.
    MINDIMENSION←DX[I];
    IF(WDY<MINDIMENSION) THEN MINDIMENSION←WDY;
    WDB← MIN(MINDIMENSION/CS); % TIME FOR SOUND TO CROSS MIN CELL DIMN.
    WDA← CMIN(WDA,WDB);
    WDB← MIN(DX[I]/U[K]);% TIME FOR MATERIAL TO CROSS CELL IN X DIRECT.
    WDA← CMIN(WDA,WDB);
    WDB← MIN(WDY/V[K]); % TIME FOR MATERIAL TO CROSS CELL IN Z DIRECTION
```

```
    WDA+ CMIN(WDA,WCB);
END;
    END;
DT+0.5×STABLFCTR×WDA;
T+T+DTLAST;
POWER(T,BER,LP);
TNEXT+(T+DT)×POWEROF10(-LP);
M1+ (TNEXT>STDT) AND BOOLEAN(17777777777776000000000(8));
M2+SHIFTL(1,M1);
M1+(M1 OR M2) AND NOT (M1 AND M2);
IF(M1 AND (BER<STDT)) THEN DT+(STDT-BER)×POWEROF10(LP);
DTLAST+DT;
CYCLE+CYCLE+1;
END; %CDT.
%
%
%
%

SUBROUTINE PH1;
  BEGIN
  CINT PASS; CREAL G;
  PREAL VA,VB,PA,PR,PB,URR,URL;
  DTH+0.;
IF RIGHTREFLECTIVE THEN
    ERROR(1)
    ELSE
    BEGIN
% SET P AT AXIS AND RIGHT
  MODE+BOOLEAN(-0);
  LOOP J+0,1,JMAX DO
  P[J]+RTL(1,,P[J]);
  MODE+SHIFTR(IMAX+1,MODE);
  LOOP J+0,1,JMAX DO P[J]+RTR(2,,P[J]);
    END;
    LOOP PASS+1,1,2 DO
    BEGIN
          MODE+TRUE;
          IF BOTTOMREFLECTIVE THEN
          BEGIN
  VB+0.;
  PB+P[0];
          END
          ELSE BEGIN
  VB+V[0];
  WCA+ GRABONE(DY[0],0);
  WCB+ GRABONE(DY[0],1);
  WCC+1./(WCA+WCB);
  PB+(P[1]×WCA+P[0]×WCB)×WCC;
  DTH+DTH+0.5×TAU×WCA×V[0]×DT×(P[0]+P[1])×WCC;
  END;
IF RIGHTREFLECTIVE THEN ERROR(2)
ELSE BEGIN
% SET U AT AXIS AND RIGHT.
  MODE+BOOLEAN(-0);
  LOOP J+0,1,JMAX DO U[J]+RTL(1,,U[J]);
  MODE+SHIFTR(IMAX+1,MODE);
  WCA+RTR(1,,RC)/RC;
  LOOP J+0,1,JMAX DO U[J]+WCA×RTR(1,,U[J]);
  END;
  MODE+NOT BOOLEAN(-1) AND (IMAX2FEN);
```

```
  WPA←1.0/(DX+RTL(1,,DX));
  WPB←RC×RTL(1,,DX]×WPA;
  WPC←RTL(1,,RC]×DX×WPA;
  WCA←GRABONE(DY[0],0);
  LOOP J←0,1,JMAX-1 DO
    BEGIN
    IF(AMX[J]≤0.] THEN ERROR(3);
    URR←U[J]×WPB+RTL(1,,U[J])×WPC;
    PR←(P[J]×RTL(1,,DX)+RTL(1,,P[J]]×DX)×WPA;
     J1←(J+1) DIV 64;
    J2←(J+1)-64×J1;
     WCB←GRABONE(DY[J1],J2);
    WCC←1.0/(WCA+WCB];
    WCD←WCA×WCC;
    WCE←WCB×WCC;
    VA←V[J]×WCE+V[J+1]×WCD;
AIX[J]←AIX[J]+P[J]×DT×(0.5×(VB-VA]×TAU+(RTR(1,,URR)-URR)×PI×WCA]/AMX[J]];
    IF(PASS=1) THEN
      BEGIN
      PA←P[J]×WCE+P[J+1]×WCD;
      G←6.356708;
      G←G/(G+GRABONE(Y[J1],J2]-0.5×WCA]];
      G←G0×G×G;
      U[J]←U[J]+TAU×WCA×(RTR(1,,PR]-PR]×DT/(DX×AMX[J]);
      V[J]←V[J]+TAU×WCA×(PB-PA)×DT/(WCA×AMX[J])-G×DT;
      PB←PA;
      END;
    VB←VA;
     WCA←WCB;
    END; % J LOOP
  J←JMAX;
  IF(AMX[J]≤0.] THEN ERROR(4);
  URR←U[J]×WPB+RTL(1,,U[J])×WPC;
    PR←(P[J]×RTL(1,,DX)+RTL(1,,P[J]]×DX)×WPA;
    IF TOPREFLECTIVE THEN ERROR(5]
     ELSE BEGIN
  VA←V[J]];
   % PA←PB; % ALREADY DONE ABOVE
  DTH←DTH-0.5×TAU×WCB×V[J]×DT×(P[J]+P[J-1]]×WCC;
     END;
AIX[J]←AIX[J]+P[J]×DT×(0.5×(VB-VA]×TAU+(RTR(1,,URR)-URR]×PI×WCA]/AMX[J]];
  IF PASS=1 THEN  U[J]←U[J]+TAU×WCA×(RTR(1,,PR)-PR]×DT/(DX×AMX[J]]];
   % DONE WITH WHOLE GRID
    END; % PASS LOOP.
  ETH←ETH+SUMROW(DTH]];
  END;%PH1
%
%
%
%
SUBROUTINE PH2;
BEGIN
     PE REAL SPECIFICENERGY,DONORTOPU,DONORTOPV,DONORTOPSPECIFICENERGY,
     VATTOPINTERFACE,DONORMASSDIVIDEDBYDY,MASSTHRUTOP,MASSTHRUBOTTOM,
     DONORRIGHTU,DONORRIGHTV,DONORRIGHTSPECIFICENERGY,
     DONORMASSDIVIDEDBYDX,UATRIGHTINTERFACE,MASSTHRURIGHT,AMXNEW,
     ONEOVERAMXNEW,UMOMENTUMTHRUTOP,VMOMENTUMTHRUTOP,UMOMENTUMTHRURIGHT,
     VMOMENTUMTHRURIGHT,ENERGYTHRURIGHT,ENERGYTHRUTOP,ENERGYTHRUBOTTOM,
     UMOMENTUMTHRUBOTTOM,VMOMENTUMTHRUBOTTOM;
     REZONETOP←FALSE;REZONERIGHT←FALSE;REZONEBOTTOM←FALSE;
     DTH←0.;
```

```
      WCA+GRABONE(DY[0],0);
      % SET AXIS BOUNDARY CONDITIONS
      MODE+BOOLEAN(-0);
      MASSTHRURIGHT+0.;
      % SET BOTTOM BOUNDARY CONDITIONS
      MODE+TRUE;
IF BOTTOMREFLECTIVE THEN
    BEGIN
      MASSTHRUBOTTOM+0.;
      ENERGYTHRUBOTTOM+0.;
      UMOMENTUMTHRUBOTTOM+0.;
      VMOMENTUMTHRUBOTTOM+0.;
    END
    ELSE BEGIN
          MASSTHRUBOTTOM+V[0]×AMX[0]×DT/WCA;
          IF(MASSTHRUBOTTOM>0.) THEN MASSTHRUBOTTOM+0.;
          IF(-MASSTHRUBOTTOM>TOZONE×AMX[0]) THEN REZONEBOTTOM+TRUE;
          ENERGYTHRUBOTTOM+MASSTHRUBOTTOM×(AIX[0]+0.5×(U[0]×U[0]+V[0]×
          V[0]));
          DTE+ENERGYTHRUBOTTOM;
          UMOMENTUMTHRUBOTTOM+MASSTHRUBOTTOM×U[0];
          VMOMENTUMTHRUBOTTOM+MASSTHRUBOTTOM×V[0];
      END;
 IF RIGHTREFLECTIVE THEN ERROR(6)
     ELSE BEGIN
      % SET RIGHT BOUNDARY CONDITIONS
      MODE+SHIFTR(IMAX+1,MODE);
      LOOP J+0,1,JMAX DO
      BEGIN
          U[J]+RTR(1,,U[J]);
          IF(U[J]<0.) THEN U[J]+-U[J];
      END;
      END;
      MODE+NOT BOOLEAN(-1) AND (IMAX≥PFN);
      LOOP J+0,1,JMAX DO
      BEGIN
          % COMPUTE QUANTITIES AT TOP INTERFACE
          SPECIFICENERGY+AIX[J]+0.5×(U[J]×U[J]+V[J]×V[J]);
          DONORTOPU+U[J];
          DONORTOPV+V[J];
          DONORTOPSPECIFICENERGY+SPECIFICENERGY;
          DONORMASSDIVIDEDBYDY+AMX[J]/WCA;
          IF J<JMAX THEN
          BEGIN
     J1+(J+1) DIV 64;
     J2+(J+1)-64×J1;
    WCB+GRABONE(DY[J1],J2);
                VATTOPINTERFACE+0.5×(V[J]+V[J+1]);
                IF (VATTOPINTERFACE<0.) THEN
                BEGIN
                      DONORMASSDIVIDEDBYDY+AMX[J]/WCB;
                      DONORTOPU+U[J+1];
                      DONORTOPV+V[J+1];
                      DONORTOPSPECIFICENERGY+AIX[J+1]+0.5×(U[J+1]×U[J+1]+
                      V[J+1]×V[J+1]);
                END;
                MASSTHRUTOP+VATTOPINTERFACE×DT×DONORMASSDIVIDEDBYDY;
          END
          ELSE BEGIN    % DO TOP ROW DIFFERENTLY.
    IF TOPREFLECTIVE THEN ERROR(7)
        ELSE BEGIN
```

```
                    VATTOPINTERFACE+V[J];
                    MASSTHRUTOP+VATTOPINTERFACE×DT×DONORMASSDIVIDEDBYDY;
                    IF(MASSTHRUTOP<0.) THEN MASSTHRUTOP+0.;
                    IF(MASSTHRUTOP>TUZONE×AMX[J]) THEN REZONETOP+TRUE;
                    DTH+DTH+MASSTHRUTOP×DONORTOPSPECIFICENERGY;
            END;
            END;
            % COMPUTE QUANTITIES AT RIGHT INTERFACE
            DONORRIGHTU+U[J];
            DONORRIGHTV+V[J];
            DONORRIGHTSPECIFICENERGY+SPECIFICENERGY;
            DONORMASSDIVIDEDBYDX+AMX[J]/(DX×RC);
            UATRIGHTINTERFACE+0.5×(U[J]+RTL(1,,U[J]));
            IF (UATRIGHTINTERFACE<0.) THEN
            BEGIN
                    DONORMASSDIVIDEDBYDX+RTL(1,,DONORMASSDIVIDEDBYDX);
                    DONORRIGHTU+RTL(1,,DONORRIGHTU);
                    DONORRIGHTV+RTL(1,,DONORRIGHTV);
                    DONORRIGHTSPECIFICENERGY+
                    RTL(1,,DONORRIGHTSPECIFICENERGY);
            END;
            MASSTHRURIGHT+UATRIGHTINTERFACE×DT×DONORMASSDIVIDEDBYDX×X;
            IF(GRABONE(MASSTHRURIGHT,IMAX)>TUZONE×GRABONE(AMX[J],IMAX))
            THEN REZONERIGHT+TRUE;
            AMXNEW+AMX[J]-MASSTHRUTOP+MASSTHRUBOTTOM-MASSTHRURIGHT+
            RTR(1,,MASSTHRURIGHT);
            ONEOVERAMXNEW+1.0/AMXNEW;
            % MOMENTUM FLUXES AT TOP AND RIGHT
            UMOMENTUMTHRUTOP+MASSTHRUTOP×DONORTOPU;
            UMOMENTUMTHRURIGHT+MASSTHRURIGHT×DONORRIGHTU;
            VMOMENTUMTHRUTOP+MASSTHRUTOP×DONORTOPV;
            VMOMENTUMTHRURIGHT+MASSTHRURIGHT×DONORRIGHTV;
            ENERGYTHRUTOP+MASSTHRUTOP×DONORTOPSPECIFICENERGY;
            ENERGYTHRURIGHT+MASSTHRURIGHT×DONORRIGHTSPECIFICENERGY;
            IF PEN=IMAX THEN DTH+DTH-ENERGYTHRURIGHT;
            % UPDATE U
            U[J]+(AMX[J]×U[J]-UMOMENTUMTHRUTOP+UMOMENTUMTHRUBOTTOM-
            UMOMENTUMTHRURIGHT+RTR(1,,UMOMENTUMTHRURIGHT))×ONEOVERAMXNEW;
            % UPDATE V
            V[J]+(AMX[J]×V[J]-VMOMENTUMTHRUTOP+VMOMENTUMTHRUBOTTOM-
            VMOMENTUMTHRURIGHT+RTR(1,,VMOMENTUMTHRURIGHT))×ONEOVERAMXNEW;
            % UPDATE AIX
            AIX[J]+(AMX[J]×SPECIFICENERGY-ENERGYTHRUTOP+ENERGYTHRUBOTTOM-
            ENERGYTHRURIGHT+RTR(1,,ENERGYTHRURIGHT)-0.5×AMXNEW×(U[J]×U[J]+
            V[J]×V[J]))×ONEOVERAMXNEW;
            % UPDATE AMX
            AMX[J]+AMXNEW;
            % STORE BOTTOM QUANTITIES FOR NEXT ROW
            UMOMENTUMTHRUBOTTOM+UMOMENTUMTHRUTOP;
            VMOMENTUMTHRUBOTTOM+VMOMENTUMTHRUTOP;
            ENERGYTHRUBOTTOM+ENERGYTHRUTOP;
            MASSTHRUBOTTOM+MASSTHRUTOP;
        END; % J LOOP
        ETH+ETH+SUMROW(DTH);
END; % FH2
%
%
%
%
SUBROUTINE REZONE (CINT J);
BEGIN
```

```
    ERROR(100);
END;
%
%
%
%
SUBROUTINE OUTPUT;
BEGIN
CU REAL TOTALENERGYNOW,RELERRNOW,ENGYCHECK,NUMBEROFCYCLES;
MODE←NOT BOOLEAN(=0) AND PENSIMAX;
WPA←0.0;
LOOP J←0,1,JMAX DO WPA←WPA+AMX[J]×(AIX[J]+0.5×(U[J]×U[J]+V[J]×V[J]));
TOTALENERGYNOW←SUMHOW(WPA);
RELERRNOW←(TOTALENERGYNOW-ETH)/ETH;
NUMBEROFCYCLES←CYCLE-CYCLASTPRINT;
IF (NUMBEROFCYCLES=0.0) THEN NUMBEROFCYCLES←1.0;
ENGYCHECK←(RELERRNOW-RELERRBEFORE)/NUMBEROFCYCLES;
RELERRBEFORE←RELERRNOW;
CYCLASTPRINT←CYCLE;
MODE←TRUE;
PUTCPR(CYCLE,Z,1);
PUTCPR(T     ,Z,2);
PUTCPR(DT    ,Z,3);
PUTCPR(DTLAST,Z,4);
PUTCPR(ETH,Z,7);
PUTCPR(STABLFCTR,Z,8);
PUTCPR(G0,Z,9);
PUTCPR(TOZONE,Z,10);
PUTCPR(0,Z,11);PUTCPR(0,Z,12);PUTCPR(0,Z,13);
IF TOPREFLECTIVE THEN PUTCPR(1,Z,11);
IF RIGHTREFLECTUVE THEN PUTCPR(1,Z,12);
IF BOTTOMREFLECTIVE THEN PUTCPR(1,Z,13);
PUTCPR(RELERRBEFORE,Z,14);
SIMWRITE(I4DISK1,Z);
%IF(RELERRNOW>MAXRELERR) THEN ERROR(200);
END; %OUTPUT.
%
%
%
%
%         MAIN PROGRAM STARTS HERE.
%
%
SHELL: OPEN(I4DISK0,0,1);
       OPEN(I4DISK1,0,1);
INPUT;
NEGATIVEMASS(1);NEGATIVEENERGY(1);
AGAIN:  CDT;
        OUTPUT;
NEGATIVEMASS(2);NEGATIVEENERGY(2);
        PH1;
NEGATIVEMASS(3);NEGATIVEENERGY(3);
        PH2;
NEGATIVEMASS(4);NEGATIVEENERGY(4);
      % AT THIS POINT SHOULD GO
      % IF(REZONETOP) THEN REZONE(1);
      % IF(REZONERIGHT) THEN REZONE(2);
      % IF(REZONEBOTTOM) THEN REZONE(3);
        GO TO AGAIN;
%
STOP:FIN:END.
```

APPENDIX XI

SHELL/OF/THE/FUTURE

```
$SUMRY DBUGA LIST ZIP
BEGIN
%
%        SHELL/OF/THE/FUTURE.
%        THIS IS IMAX*JMAX SHELL WRITTEN FOR OUT OF CORE JOBS.
%        R.W. CLEMENS 14 MAY 71, REVISED 24 AUG 71.
%
%        CURRENTLY DIMENSIONED FOR A MAX OF 638*1280 IN A MAXIMUM OF
%        50 ROW BLOCKS.
%        NB A GIVEN I MAY (WILL) BE MORE THAN ONE OF THESE ROWS.
%
%        I AND J REFER TO THE UPPER RIGHT CORNER OF A CELL.
%        THERE IS NO DUMMY ROW AT THE TOP OR BOTTOM OF THE GRID.
%        THERE IS A DUMMY COLUMN AT THE LEFT AND RIGHT OF THE GRID.
%
CINT IMAX,JMAX,I,J,K,J1,J2,IBLK,IBLKP1,IROWS,IROWSM1,JSTRIPS,JBOT,
     JMAXPERSTRIP,JPL,JPN,JB,JC,JJ,IROWS2,IROWS3,IROWS4,JSTRIPSP1;
CREAL PROB,CYCLE,T,DT,WCA,WCB,WCC,WCD,WCE,WCF,PI,ETH,STABLFCTR,
     RELERRBEFORE,CYCLASTPRINT,MAXRELERR,GO,TOZONE,TOUMP,CYCLESTOP,
     WDA,TOTALENERGYNOW;
PINT IR,IR2,IL;
PREAL Z,CS,WPA,WPB,WPC,DTH,STDT,PA,VA,ZSALUTE;
BOOLEAN BOTTOMREFLECTIVE,RIGHTREFLECTIVE,TOPREFLECTIVE,MODE1,
     REZONETOP,REZONERIGHT,REZONEBOTTOM,STOPTHISCYCLE;
BOOLEAN OLDMODE;   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
BOOLEAN VECTOR SW[6];
LABEL SHELL,AGAIN,STOP,FIN;
%
%        U, V, AMX, AIX, AND P MUST BE DIMENSIONED TO AT LEAST
%             2*(BLOCK SIZE)-1.
%
PREAL VECTOR U,V,AMX,AIX,P[99];
%
%        THE FOLLOWING ITEMS MUST BE DIMENSIONED TO AT LEAST
%             5*(BLOCK SIZE)-1.
%
PREAL VECTOR LAST,NEXT[249];
%
%        THE FOLLOWING ITEMS MUST BE DIMENSIONED AT LEAST (IMAX+1)/64.
%
PREAL VECTOR X,DX,RC,TAU,VB,PR,PB,URR,WP1,WP2,WP3,
     MASSTHRUBOTTOM,ENERGYTHRUBOTTOM,UMOMENTUMTHRUBOTTOM,ENERGYTHRUTOP,
     UMOMENTUMTHRUTOP,VMOMENTUMTHRUTOP,UMOMENTUMTHRURIGHT,
     VMOMENTUMTHRURIGHT,ENERGYTHRURIGHT,DONORRIGHTU,DONORRIGHTV,
     DONORRIGHTSPECIFICENERGY,MASSTHRURIGHT,MASSTHRUTOP,
     CONORMASSDIVIDEDBYDX,VMOMENTUMTHRUBOTTOM,SPECIFICENERGY[9];
BOOLEAN VECTOR IMODE,IMODEL[9];
%
%        THE FOLLOWING ITEMS MUST BE DIMENSIONED AT LEAST JMAX (SIC).
%
CREAL VECTOR Y,DY[1280];
%
%
%
%I FILE I4INOUT   (BUFFERS=0,ACCESS=RANDOM,PHYSICALRECORDSIZE=250 ROWS,         11111
%I          TITLE=+SHELL/I4INOUT.+);                                          11111
%I FILE I4SCR     (BUFFERS=0,ACCESS=RANDOM,PHYSICALRECORDSIZE=250 ROWS,         11111
%I          TITLE=+SHELL/I4SCR.+);                                            11111
%I FILE I4SALUTE (BUFFERS=0,ACCESS=RANDOM,PHYSICALRECORDSIZE=1 ROW,            11111
%I          TITLE=+SHELL/I4SALUTE.+);                                         11111
```

```
FILE I4DISK0=+SHELL/I4CLAM/OUT60BY15+ (250 ROWS FULL);              11111
FILE I4DISK7=+SHELL/I4SCR+   (250 ROWS FULL);                       11111
FILE LINE (1 ROW);                                                  11111
%
%
%        FOLLOWING ARE ALL SUBROUTINES.
%
%
PREAL SUBROUTINE ALOG AS RGA (PREAL X AS RGA);
BEGIN
$ INCINS=100, ASKRATCH=18
$$TAPE2=GLYPNIR/GLN SERIAL;
END; %LN.
%
%
%
%
PREAL SUBROUTINE EXP AS RGA (PREAL X AS RGA);
BEGIN
$ INCINS=100, ASKRATCH=17
$$TAPE2=GLYPNIR/GEXP SERIAL;
END; %EXP.
%
%
%
%
PREAL SUBROUTINE SQRT AS RGA (PREAL X AS RGA);
BEGIN
$ INCINS=110, ASKRATCH=14
$$TAPE2=GLYPNIR/GSQRT SERIAL;
END; %SQRT.
%
%
%
%
PREAL SUBROUTINE ALOG10(PREAL X);
ALOG10=0.4342944819032518*ALOG(X);
%
%
%
%
PREAL SUBROUTINE POWERCF10 (PREAL N);
POWEROF10=EXP(2.302585092994046*N); % POWEROF10=10.**N.
%
%
%
%
SUBROUTINE POWER(PREAL X,PREAL OUT XX,PINT OUT N);
BEGIN
      N=TRUNCATE(ALOG10(ABS(X)));
      IF(N<0) THEN N=N-1;
      XX=X*POWEROF10(-N);
END; %POWER.
%
%
%
%
SUBROUTINE ERROR(CINT J);
BEGIN
%I    EXIT(J);                                                      11111
```

```
      SIMWRITE(LINE[PAGE],*ERROR =*,J);                                                         11111
      WCA=0.;                                                                                   11111
      WCA=1.0+1.0/WCA; %ABORT.                                                                  11111
END; %ERROR.
%
%
%
%
SUBROUTINE DATA;
BEGIN
           CODE
USE        STDT,PI,IR,IR2,IL;
           BEGIN
HEREIAM    SET ;
           ORG  PI;
           DATA 3.141592653589793;
           ORG  STDT;
           DATA 1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.3,2.5,2.6,
                2.8,3.0,3.2,3.4,3.6,3.8,4.0,4.2,4.5,4.8,5.0,5.3,5.6,6.0,
                6.5,7.0,7.5,8.0,8.5,9.0,9.5,10.0;
           ORG  IR;
           DATA (0)63,-1;
           ORG  IR2;
           DATA (0)62,(-1)2;
           ORG  IL;
           DATA 1,(0)63;
           ORG  HEREIAM;
           END  CODE;
END; %DATA.
%
%
%
%
SUBROUTINE ES(CINT K);
BEGIN
%
%         THIS IS 11 MAY 69 VERSION OF AIR.
%         IT WILL GIVE SLIGHTLY DIFFERENT ANSWERS THAN THAT CODE
%         AS HERE WE COMPUTE THE EXPONENTIAL SWITCHS IN ALL CASES.
%
%         ES USES RHO [IN CS ARRAY], AND AIX[K].
%         IT RETURNS UPDATED P[K] AND CS.
%
      FREAL E,LNEROVROZ,FO,FON,FN,WS,BETA,FE,GMONE;
      FREAL RHO AS CS;
      E=1.0+-10*AIX[K];
      LNEROVROZ=ALOG(773.39520495*RHO);
      WS=1.0/(1.0+EXP((-8.5-0.15504314*LNEROVROZ+E)*EXP(-0.05*LNEROVROZ
           +0.02531780798)));
      FO =WS *EXP(-0.22421524664*E);
      FON=(1.0-WS )*EXP(-0.15082956259*E);
      BETA=0.;
      IF(E>1.)THEN BETA= (0.0069487*WS+0.0138974)*ALOG(E);
      WS=1.0/(1.0+EXP((-E+EXP(0.0157*LNEROVROZ+3.806662489))*EXP(-0.085
           *LNEROVROZ-1.38629436)));
      FN=WS*EXP(-0.039215686275*E);
      BETA=BETA*(1.0-WS)+0.045*WS;
      WPA=30.0+3.474356*LNEROVROZ;
      IF WPA <6.0 THEN WPA=6.0;
      WPA=(160.0-E)/WPA;
```

```
      FE=1.0/(1.0+EXP(WPA));
      GMONE=(0.161+0.255*FO+0.280*FON+0.137*FN+0.05*FE)*EXP(BETA
            *LNEROVROZ);
      P[K]=GMONE*RHO*AIX[K];
      CS=SQRT((1.0+GMONE)*P[K]/RHO);
END; %ES.
%
%
%
%
SUBROUTINE CDT;
BEGIN
      PREAL MINDIMENSION; CREAL WDB;
      LOOP I=0,1,IBLK DO BEGIN
            K=I;
            LOOP JJ=1,1,JMAXPERSTRIP DO BEGIN
                  J=JJ+JBOT;
                  MODE=IMODE[I];
                  CS=AMX[K]/(DY[J]*TAU[I]);
                  ES(K);
                  MINDIMENSION=DX[I];
                  IF DY[J]<MINDIMENSION THEN MINDIMENSION=DY[J];
                  WDB=MAX(CS/MINDIMENSION);  % INV TIME FOR SOUND TO CROSS
                  IF WDB>WDA THEN WDA = WDB; % MINIMUM CELL DIMENSION.
                  WDB=MAX(ABS(U[K])/DX[I]);  % TIME FOR MATERIAL TO CROSS
                  IF WDB>WDA THEN WDA = WDB; % CELL IN X DIRECTION.
                  WDB=MAX(ABS(V[K])/DY[J]);  % TIME FOR MATERIAL TO CROSS
                  IF WDB>WDA THEN WDA = WDB; % CELL IN Y DIRECTION.
                  K=K+IBLKP1;
            END; %JLOOP.
      END; %ILOOP.
MODE=TRUE;
SIMWRITE(LINE,+CDT-WDA+,WDA);
END; % CDT.
%
%
%
SUBROUTINE CHOOSEDT;
BEGIN
      CREAL REG,TNEXT; CINT LP; BOOLEAN M1,M2,EVERYSTDTIME;
      %
      %   THE BOOLEAN EVERYSTDTIME CHOOSES ONE OF TWO OPTIONS FOR
      %   COMPUTING STANDARD TIMES.  THE TWO OPTIONS WILL DIFFER ONLY
      %   DURING THE FIRST FEW CYCLES OF A CALCULATION WHEN IT IS
      %   POSSIBLE FOR T+DT TO VAULT ACROSS SEVERAL STANDARD TIMES.
      %
      EVERYSTDTIME=TRUE;
      DT=0.5*STABLFCTR/WDA;
      WDA=1.0+-20;
      T=T+DT;
      POWER(T,REG,LP);
      TNEXT=(T+DT)*POWEROF10(-LP);
      IF EVERYSTDTIME THEN BEGIN
            % THEN CLAUSE FINDS EVERY STANDARD TIME.
            M1=(REG≤STDT) OR BOOLEAN(1777777777(8));
            M2=SHIFTR(1,M1);
            M1=(M1 OR M2) AND NOT (M1 AND M2);
            IF M1 AND (TNEXT>STDT) THEN BEGIN
                  DT=(STDT-REG)*POWEROF10(LP);
MODE=TRUE;
```

```
SIMWRITE(LINE,+DT+,DT);
                TOTALENERGYNOW=0;
                TDUMP=CYCLE+2;
%%%
%%%
%%%      HERE PULSE OSK TO LET IT KNOW THAT A DUMP IS COMING AT THE END
%%%      OF THIS CYCLE.  THIS SHOULD GIVE IT TIME TO LOAD THE CORRECT
%%%      STRIP TO LASER STORE.
%%%
%%%      FORMAT OF OSK STATEMENT IS OF YET UNDEFINED.
%%%
%%%
           END;
      END
      ELSE BEGIN
           % ELSE CLAUSE FINDS THE LARGEST STANDARD TIME LESS THAN T+DT.
           M1= (TNEXT≥STDT) AND BOOLEAN(177777777777600000000(8));
           M2=SHIFTL(1,M1);
           M1=(M1 OR M2) AND NOT (M1 AND M2);
           IF M1 AND (REG<STDT) THEN BEGIN
                DT=(STDT-REG)*POWEROF10(LP);
                TOTALENERGYNOW=0;
                TDUMP=CYCLE+2;
%%%
%%%
%%%      HERE PULSE OSK TO LET IT KNOW THAT A DUMP IS COMING AT THE END
%%%      OF THIS CYCLE.  THIS SHOULD GIVE IT TIME TO LOAD THE CORRECT
%%%      STRIP TO LASER STORE.
%%%
%%%      FORMAT OF OSK STATEMENT IS OF YET UNDEFINED.
%%%
%%%
           END;
      END;
      CYCLE=CYCLE+1;
      STOPTHISCYCLE=FALSE;
      IF CYCLE≥CYCLESTOP OR SW[4] THEN STOPTHISCYCLE=TRUE;
END; %CHOOSEDT.
%
%
%
%
SUBROUTINE SETDXDYETC;
BEGIN
      MODE=NOT BOOLEAN(-0);
      LOOP I=0,1,IBLK DO BEGIN
           IMODE[I]=MODE;
           MODE=TRUE;
           IMODEL[I]=MODE;
      END;
      IMODE[IBLK]=IMODE[IBLK] AND IMAX≥64*IBLK+PEN;
      IMODEL[IBLK]=IMODEL[IBLK] AND IMAX≥64*IBLK+PEN;
      LOOP I=0,1·IBLK DO BEGIN
           MODE=¬MODE[I];
           WPA=RTR(1,,X[I+IR]);
           RC[I]=0.5*(X[I]+WPA);
           DX[I]=X[I]-WPA;
           TAU[I]=PI*(X[I]*X[I]-WPA*WPA);
      END;
      MODE=BOOLEAN(-0);
```

```
     DX[0]=RTL(1,,DX[0]);
     RC[0]=-RTL(1,,RC[0]);
     MODE=REVR(IMAX+1-64*IBLK,MODE);
     DX[IBLK]=RTR(2,,DX[IBLK+IR2]);
     RC[IBLK]=RTR(1,,X[IBLK+IR])+0.5*DX[IBLK];
     LOOP I=0,1,IBLK DO BEGIN
          MODE=IMODEL[I];
          WPA=RTL(1,,DX[I+IL]);
          WP1[I]=1.0/(DX[I]+WPA);
          WP2[I]=WPA*RC[I]*WP1[I];
          WP3[I]=RTL(1,,RC[I+IL])*DX[I]*WP1[I];
     END;
     MODE=TRUE;
     LOOP J=1,1,JMAX DO DY[J]=Y[J]-Y[J-1]; % DY[0] IS NOT USED IN CODE.
END; %SETDXDYETC.
%
%
%
%
SUBROUTINE INPUT;
BEGIN
%
%
%        THE CURRENTLY ASSIGNED LOCATIONS IN THE Z-BLOCK ARE
%
%         0.  PROBLEM NO.
%         1.  CYCLE.
%         2.  TIME.
%         3.  DT.
%         4.
%         5.  IMAX.
%         6.  JMAX.  JMAX MUST BE A MULTIPLE OF JMAXPERSTRIP.
%         7.  ETH.
%         8.  STABLFCTR, SHOULD BE 0.5.
%         9.  G0,  GRAVITY AT THE EARTHS SURFACE.
%        10.  TOZONE,  PERCENT MASS LEAVING A BOUNDARY CELL TO TRIGGER
%             A REZONE.
%        11.  TOP REFLECTIVE, NONZERO FOR REFLECTIVE.
%        12.  RIGHT REFLECTIVE, NONZERO FOR REFLECTIVE.
%        13.  BOTTOM REFLECTIVE, NONZERO FOR REFLECTIVE.
%        14.  RELATIVE ERROR AT LAST PRINT.
%        15.  MAXIMUM RELATIVE ERROR ALLOWED.
%        16.  CYCLE STOP.
%        17.  JMAX PER HORIZONTAL STRIP PROCESSED.
%
%
      MODE=TRUE;
 SIMWRITE(LINE,+ENTER INPUT+);                                              11111
 %I   READ(I4INOUT[0],NEXT);                                                11111
 %I   WAIT;                                                                 11111
      SIMREAD(I4DISK0[0],NEXT); % READS Z, X, AND Y.  CLOSE PACKED.         11111
      Z=NEXT[0];
      PROB =GRABONE(Z,0);
      CYCLE=GRABONE(Z,1);
      CYCLASTPRINT=CYCLE;
      T=GRABONE(Z,2);
      DT=GRABONE(Z,3);
      IMAX =GRABONE(Z,5);
      JMAX =GRABONE(Z,6);
      ETH=GRABONE(Z,7);
```

```
      STABLFCTR=GRABONE(Z,8);
      IF(STABLFCTR≤0.) THEN STABLFCTR=1.;
      G0=GRABONE(Z,9);
      TOZONE=GRABONE(Z,10);
      TOPREFLECTIVE = FALSE;
      RIGHTREFLECTIVE = FALSE;
      BOTTOMREFLECTIVE = FALSE;
      IF(GRABONE(Z,11)≠0.) THEN TOPREFLECTIVE=TRUE;
      IF(GRABONE(Z,12)≠0.) THEN RIGHTREFLECTIVE=TRUE;
      IF(GRABONE(Z,13)≠0.) THEN BOTTOMREFLECTIVE=TRUE;
      RELERRBEFORE=GRABONE(Z,14);
      MAXRELERR=GRABONE(Z,15);
      CYCLESTOP=GRABONE(Z,16);
      JMAXPERSTRIP= GRABONE(Z,17);
      JSTRIPS=JMAX DIV JMAXPERSTRIP;
      JSTRIPSP1=JSTRIPS+1;
      IBLK=(IMAX+1) DIV 64;
      IBLKP1=IBLK+1;
      IROWS=IBLKP1*JMAXPERSTRIP;
      IROWSM1=IROWS-1;
      IROWS2=IROWS+IROWS;
      IROWS3=IROWS2+IROWS;
      IROWS4=IROWS3+IROWS;
      LOOP I=0,1,IBLK DO X[I]=NEXT[I+1];
      LOOP J=0,1,JMAX DO BEGIN
           J1=J DIV 64;
           J2=J-64*J1;
           Y[J]=GRABONE(NEXT[J1+IBLK+2],J2);
      END;
      T=T-DT;
      CYCLE=CYCLE-1;
SIMWRITE(LINE,+TO SET+);
      SETJXDYETC;
SIMWRITE(LINE,+BACK FROM SET+);
%I    READ(I4INOUT[1],LAST);                                              11111
%I    READ(I4INOUT[2],NEXT);                                              11111
%I    WAIT(I4INOUT[1]);                                                   11111
      SIMREAD(I4DISK0[1],LAST);                                           11111
      SIMREAD(I4DISK0[2],NEXT);                                           11111
      LOOP JC=0,1,IROWSM1 DO BEGIN
           U  [JC]=LAST[JC];
           V  [JC]=LAST[JC+IROWS];
           AMX[JC]=LAST[JC+IROWS2];
           AIX[JC]=LAST[JC+IROWS3];
           P  [JC]=LAST[JC+IROWS4];
      END;
      JPL=0;
      JPN=2;
      WDA=1.0+-20;
      JBOT=0;
      LOOP JB=1,1,JSTRIPS DO BEGIN
SIMWRITE(LINE,+JB ETC+,JB,JBOT,JPL,JPN,AIX[0],AIX[1],AIX[2],AIX[3]);
           CDT;
%I         WAIT(I4SCR[JPL]);                                              11111
%I         WAIT(I4INOUT[JPN]);                                            11111
           JPL=JPL+1;
           JPN=JPN+1;
           IF JPL=JSTRIPSP1 THEN JPL=1;
           IF JPN=JSTRIPSP1 THEN JPN=1;
           LOOP JC=0,1,IROWSM1 DO BEGIN
```

```
                LAST[JC        ]=U  [JC];
                LAST[JC+IROWS  ]=V  [JC];
                LAST[JC+IROWS2]=AMX[JC];
                LAST[JC+IROWS3]=AIX[JC];
                LAST[JC+IROWS4]=P  [JC];
                %
                U   [JC]=NEXT[JC];
                V   [JC]=NEXT[JC+IROWS ];
                AMX[JC]=NEXT[JC+IROWS2];
                AIX[JC]=NEXT[JC+IROWS3];
                P   [JC]=NEXT[JC+IROWS4];
            END; %JC LOOP.
%I          WRITE(I4SCR[JPL],LAST);                                                   11111
%I          READ(I4INOUT[JPN],NEXT);                                                  11111
            SIMWRITE(I4DISK7[JPL],LAST);                                              11111
            SIMREAD(I4DISK0[JPN],NEXT);                                               11111
            JBOT=JBOT+JMAXPERSTRIP;
        END; %JB LOOP.
END; %INPUT.
%
%
%
%
SUBROUTINE PH1;
BEGIN
        CINT PASS,JM; CREAL G;
        MODE=TRUE;
SIMWRITE(LINE,+INTO PH1+);
        DTH=0;
        URR[0]=0.;
        % SET P AT AXIS.
        MODE=BOOLEAN(-0);
        LOOP K=0,IBLKP1,IROWSM1 DO P[K]=RTL(1,,P[K]);
        IF RIGHTREFLECTIVE THEN ERROR(1)
        ELSE BEGIN
            % SET P AT RIGHT.
            MODE=REVR(IMAX+1-64*IBLK,MODE);
            LOOP K=IBLK,IBLKP1,IROWSM1 DO P[K]=RTR(2,,P[K+IR2]);
        END;
        LOOP PASS=1,1,2 DO BEGIN
            MODE=TRUE;
            IF JB=0 THEN IF BOTTOMREFLECTIVE THEN LOOP I=0,1,IBLK DO BEGIN
                VB[I]=0.;
                PB[I]=P[I];
            END
            ELSE BEGIN
                WCA=DY[1];
                WCB=DY[2];
                WCC=1.0/(WCA+WCB);
                LOOP I=0,1,IBLK DO BEGIN
                    MODE=IMODE[I];
                    VB[I]=V[I];
                    PB[I]=(P[I+IBLKP1]*WCA+P[I]*WCB)*WCC;
                    DTH=DTH+0.5*WCA*WCC*DT*TAU[I]*V[I]*(P[I]
                        +P[I+IBLKP1]);
                END;
            END;
            % SET U AT AXIS.
            MODE=BOOLEAN(-0);
            LOOP K=0,IBLKP1,IROWSM1 DO U[K]=-RTL(1,,U[K+IL]);
```

```
MODE=REVR(IMAX+1-64*IBLK,MODE);
% SET U AT RIGHT.
IF RIGHTREFLECTIVE THEN ERROR(2)
ELSE BEGIN
      WCA=RTR(1,,RC[IBLK+IR])/RC[IBLK];
      LOOP K=IBLK,IBLKP1,IROWSM1 DO U[K]=WCA*RTR(1,,U[K+IR]);
END;
%
% HAVE SET ALL BOUNDARY CONDITIONS BOTTOM,LEFT,RIGHT.
% NOW DO SOME PHYSICS.
%
WCA=DY[JBOT+1];
K=0;
IF JB=JSTRIPS THEN JM=JMAXPERSTRIP-1 ELSE JM=JMAXPERSTRIP;
LOOP JJ=1,1,JM DO BEGIN
      J=JJ+JBOT;
      WCB=DY[J+1];
      WCC=1.0/(WCA+WCB);
      WCD=WCA*WCC;
      WCE=WCB*WCC;
      WCF=Y[J];
      LOOP I=0,1,IBLK DO BEGIN
            MODE=IMODE[I];
            IF(AMX[K]≤0.) THEN ERROR(3);
            URR[I]=U[K]*WP2[I]+RTL(1,,U[K+IL])*WP3[I];
            MODE=IMODEL[I];
            PR[I]=(P[K]*RTL(1,,DX[I+IL])+RTL(1,,P[K+IL])*DX[I])
                  *WP1[I];
            K=K+1;
      END; %ILOOP1.
      K=K-IBLKP1;
      LOOP I=0,1,IBLK DO BEGIN
            MODE=IMODE[I];
            VA=V[K]*WCE+V[K+IBLKP1]*WCD;
            AIX[K]=AIX[K]+P[K]*DT*(0.5*(VB[I]-VA)*TAU[I]
                  +(RTR(1,,URR[I+IR])-URR[I])*PI*WCA)/AMX[K];
            IF(PASS=1) THEN BEGIN
                  PA=P[K]*WCE+P[K+IBLKP1]*WCD;
                  G=6.3567+8;
                  G=G/(G+WCF-0.5*WCA);
                  G=G0*G*G;
                  U[K]=U[K]+TAU[I]*WCA*(RTR(1,,PR[I+IR])-PR[I])
                        *DT/(DX[I]*AMX[K]);
                  V[K]=V[K]+TAU[I]*(PB[I]-PA)*DT/AMX[K]-G*DT;
                  PB[I]=PA;
            END;
            VB[I]=VA;
            K=K+1;
      END; %ILOOP2.
      WCA=WCB;
END; % J LOOP
IF JB=JSTRIPS THEN BEGIN %TOP ROW CODE.
      LOOP I=0,1,IBLK DO BEGIN
            MODE=IMODE[I];
            IF(AMX[K]≤0.) THEN ERROR(4);
            URR[I]=U[K]*WP2[I]+RTL(1,,U[K+IL])*WP3[I];
            MODE=IMODEL[I];
            PR[I]=(P[K]*RTL(1,,DX[I+IL])+RTL(1,,P[K+IL])*DX[I])
                  *WP1[I];
            K=K+1;
```

```
                END; %ILOOP3.
                K=K-IBLKP1;
                IF TOPREFLECTIVE THEN ERROR(5)
                ELSE BEGIN
                     LOOP I=0,1,IBLK DO BEGIN
                          MODE=IMODE[I];
                          DTH=DTH-0.5*TAU[I]*WCB*V[K]*DT*(P[K]
                               -P[K-IBLKP1])*WCC;
                          K=K+1;
                     END;
                     K=K-IBLKP1;
                END;
                LOOP I=0,1,IBLK DO BEGIN
                     MODE=IMODE[I];
                     VA=V[K];
                     AIX[K]=AIX[K]+P[K]*DT*(0.5*(VB[I]-VA)*TAU[I]
                          +(RTR(1,,URR[I+IR])-URR[I])*PI*WCA)/AMX[K];
                     IF (PASS=1) THEN U[K]=U[K]+TAU[I]*WCA
                          *(RTR(1,,PR[I+IR])-PR[I])*DT/(DX[I]*AMX[K]);
                     K=K+1;
                END;
           END; %TOP ROW CODE.
      END; % PASS LOOP
      MODE=TRUE;
      ETH=ETH+ROWSUM(DTH);
END; %PH1.
%
%
%
%
SUBROUTINE PH2;
BEGIN
      CINT  K1;
      FREAL VATTOPINTERFACE,DONORMASSDIVIDEDBYDY,
           DONORTOPU,DONORTOPV,DONORTOPSPECIFICENERGY,
           ONEOVERAMXNEW,UATRIGHTINTERFACE,AMXNEW;
      REZONETOP=FALSE;REZONERIGHT=FALSE;REZONEBOTTOM=FALSE;
      MODE=TRUE;
SIMWRITE(LINE,+INTO PH2+);
      DTH=0.;
      WCA=DY[JBOT+1];
      % SET AXIS BOUNDARY CONDITIONS.
      MASSTHRURIGHT[0]=0.;
      % SET BOUNDARY CONDITIONS AT RIGHT.
      IF RIGHTREFLECTIVE THEN ERROR(6)
      ELSE BEGIN
           MODE=REVR(IMAX+1-64*IBLK,BOOLEAN(-0));
           LOOP K=IBLK,IBLKP1,IROWSM1 DO BEGIN
                U[K]=RTR(1,,U[K+IR]);
                IF (U[K]<0.) THEN U[K]=-U[K];
           END;
      END;
      % SET BOTTOM BOUNDARY CONDITION.
      IF JB=0 THEN IF BOTTOMREFLECTIVE THEN LOOP I=0,1,IBLK DO BEGIN
           MODE=TRUE;
           MASSTHRUBOTTOM[I]=0.;
           ENERGYTHRUBOTTOM[I]=0.;
           UMOMENTUMTHRUBOTTOM[I]=0.;
           VMOMENTUMTHRUBOTTOM[I]=0.;
      END
```

```
ELSE LOOP I=0,1,IBLK DO BEGIN
     MODE=IMODE[I];
     MASSTHRUBOTTOM[I]=V[I]*AMX[I]*DT/WCA;
     IF(MASSTHRUBOTTOM[I]>0.) THEN MASSTHRUBOTTOM[I]=0.;
     IF(-MASSTHRUBOTTOM[I]>TOZONE*AMX[I]) THEN REZONEBOTTOM=TRUE;
     ENERGYTHRUBOTTOM[I]=MASSTHRUBOTTOM[I]*(AIX[I]+0.5*(U[I]*U[I]
          +V[I]*V[I]));
     OTH=OTH+ENERGYTHRUBOTTOM[I];
     UMOMENTUMTHRUBOTTOM[I]=MASSTHRUBOTTOM[I]*U[I];
     VMOMENTUMTHRUBOTTOM[I]=MASSTHRUBOTTOM[I]*V[I];
END;
K=C;
LOOP JJ=1,1,JMAXPERSTRIP DO BEGIN
     J= JJ+JBOT;
     % COMPUTE QUANTITIES AT TOP INTERFACE OF CELL.
     IF J<JMAX THEN WCB=DY[J+1];
     LOOP I=0,1,IBLK DO BEGIN
          MODE=IMODE[I];
          SPECIFICENERGY[I]=AIX[K]+0.5*(U[K]*U[K]+V[K]*V[K]);
          DONORRIGHTSPECIFICENERGY[I]=SPECIFICENERGY[I];
          DONORMASSDIVIDEDBYDX[I]=AMX[K]/(DX[I]*RC[I]);
          DONORRIGHTU[I]=U[K];
          DONORRIGHTV[I]=V[K];
          K=K+1;
     END; %ILOOP1.
     K=K-IBLKP1;
     LOOP I=0,1,IBLK DO BEGIN
          MODE=IMODE[I];
          DONORTOPU=U[K];
          DONORTOPV=V[K];
          DONORTOPSPECIFICENERGY=SPECIFICENERGY[I];
          DONORMASSDIVIDEDBYDY=AMX[K]/WCA;
          IF J<JMAX THEN BEGIN
               K1=K+IBLKP1;
               VATTOPINTERFACE=0.5*(V[K]+V[K1]);
               VATTOPINTERFACE=VATTOPINTERFACE/(1.0+(V[K1]-V[K])
                    *DT/WCA); % THIS GIVES END OF SLUG VELOCITY.
               IF (VATTOPINTERFACE<0.) THEN BEGIN
                    DONORMASSDIVIDEDBYDY=AMX[K1]/WCB;
                    DONORTOPU=U[K1];
                    DONORTOPV=V[K1];
                    DONORTOPSPECIFICENERGY=AIX[K1]
                         +0.5*(U[K1]*U[K1]+V[K1]*V[K1]);
               END;
               MASSTHRUTOP[I]=VATTOPINTERFACE*DT
                    *DONORMASSDIVIDEDBYDY;
          END
          ELSE BEGIN    % DO TOP ROW DIFFERENTLY.
               IF TOPREFLECTIVE THEN ERROR(7)
               ELSE BEGIN
                    VATTOPINTERFACE=V[K];
                    MASSTHRUTOP[I]=VATTOPINTERFACE*DT
                         *DONORMASSDIVIDEDBYDY;
                    IF(MASSTHRUTOP[I]<0.) THEN MASSTHRUTOP[I]=0.;
                    IF MASSTHRUTOP[I]>TOZONE*AMX[K] THEN REZONETOP
                         =TRUE;
                    DTH=DTH+MASSTHRUTOP[I]*DONORTOPSPECIFICENERGY;
               END;
          END;
          % COMPUTE QUANTITIES AT RIGHT INTERFACE OF CELLS.
```

```
        UATRIGHTINTERFACE=0.5*(U[K]+RTL(1,,U[K+IL]));
        UATRIGHTINTERFACE=UATRIGHTINTERFACE/(1.0+(RTL(1,,U[K+IL])
             -U[K])*DT/DX[I]); % THIS GIVES REAR OF SLUG VELOCITY
        IF (UATRIGHTINTERFACE<0.) THEN BEGIN
             DONORMASSDIVIDEDBYDX[I]=RTL(1,,DONORMASSDIVIDEDBYDX
                  [I+IL]);
             DONORRIGHTU[I]=RTL(1,,DONORRIGHTU[I+IL]);
             DONORRIGHTV[I]=RTL(1,,DONORRIGHTV[I+IL]);
             DONORRIGHTSPECIFICENERGY[I]=
                  RTL(1,,DONORRIGHTSPECIFICENERGY[I+IL]);
        END;
        MASSTHRURIGHT[I]=UATRIGHTINTERFACE*DT
             *DONORMASSDIVIDEDBYDX[I]*X[I];
        % MOMENTUM FLUXES AT TOP AND RIGHT
        UMOMENTUMTHRUTOP[I]=MASSTHRUTOP[I]*DONORTOPU;
        UMOMENTUMTHRURIGHT[I]=MASSTHRURIGHT[I]*DONORRIGHTU[I];
        VMOMENTUMTHRUTOP[I]=MASSTHRUTOP[I]*DONORTOPV;
        VMOMENTUMTHRURIGHT[I]=MASSTHRURIGHT[I]*DONORRIGHTV[I];
        ENERGYTHRUTOP[I]=MASSTHRUTOP[I]*DONORTOPSPECIFICENERGY;
        ENERGYTHRURIGHT[I]=MASSTHRURIGHT[I]
             *DONORRIGHTSPECIFICENERGY[I];
        K=K+1;
   END; %ILOOP2.
   K=K-IBLKP1;
   MODE=TRUE;
   J1=IMAX DIV 64;
   J2=IMAX-64*J1;
   IF (GRABONE(MASSTHRURIGHT[J1],J2)>
        TOZONE*GRABONE(AMX[K+J1],J2)) THEN REZONERIGHT=TRUE;
   IF (PEN=J2) THEN DTH=DTH-ENERGYTHRURIGHT[J1];
   LOOP I=0,1,IBLK DO BEGIN
        MODE=IMODE[I];
        AMXNEW=AMX[K]-MASSTHRUTOP[I]+MASSTHRUBOTTOM[I]
             -MASSTHRURIGHT[I]+RTR(1,,MASSTHRURIGHT[I+IR]);
        ONEOVERAMXNEW=1.0/AMXNEW;
        % UPDATE U
        U[K]=(AMX[K]*U[K]-UMOMENTUMTHRUTOP[I]
             +UMOMENTUMTHRUBOTTOM[I]-UMOMENTUMTHRURIGHT[I]
             +RTR(1,,UMOMENTUMTHRURIGHT[I+IR]))*ONEOVERAMXNEW;
        % UPDATE V
        V[K]=(AMX[K]*V[K]-VMOMENTUMTHRUTOP[I]
             +VMOMENTUMTHRUBOTTOM[I]-VMOMENTUMTHRURIGHT[I]
             +RTR(1,,VMOMENTUMTHRURIGHT[I+IR]))*ONEOVERAMXNEW;
        % UPDATE AIX
        AIX[K]=(AMX[K]*SPECIFICENERGY[I]-ENERGYTHRUTOP[I]
             +ENERGYTHRUBOTTOM[I]-ENERGYTHRURIGHT[I]
             +RTR(1,,ENERGYTHRURIGHT[I+IR])-0.5*AMXNEW
             *(U[K]*U[K]+V[K]*V[K]))*ONEOVERAMXNEW;
        % UPDATE AMX
        AMX[K]=AMXNEW;
        % STORE BOTTOM QUANTITIES FOR NEXT ROW
        UMOMENTUMTHRUBOTTOM[I]=UMOMENTUMTHRUTOP[I];
        VMOMENTUMTHRUBOTTOM[I]=VMOMENTUMTHRUTOP[I];
        ENERGYTHRUBOTTOM[I]=ENERGYTHRUTOP[I];
        MASSTHRUBOTTOM[I]=MASSTHRUTOP[I];
        K=K+1;
   END; %ILOOP3.
END; %JLOOP.
MODE=TRUE;
ETH=ETH+ROWSUM(DTH);
```

(

```
END; %PF2.
%
%
%
%
SUBROUTINE SUMTOTALENERGY;
BEGIN
     MODE=TRUE;
          WPA=0.;
          LOOP I=0,1,IBLK DO BEGIN
               MODE=IMODE[I];
               K=I;
               LOOP J=1,1,JMAX DO BEGIN
                    WPA=WPA+AMX[K]*(AIX[K]+0.5*(U[K]*U[K]+V[K]*V[K]));
                    K=K+IBLKP1;
               END; % J LOOP
          END; % I LOOP
          MODE=TRUE;
          TOTALENERGYNOW=TOTALENERGYNOW+ROWSUM(WPA);
END; %SUMTOTALENERGY.
%
%
%
SUBROUTINE OUTPUT;
BEGIN
     CREAL RELERRNOW,ENGYCHECK,NUMBEROFCYCLES;
     MODE=TRUE;
SIMWRITE(LINE,+INTO OUTPUT+);
%I   READ(I4SALUTE[0],ZSALUTE);                                        11111
%I   WAIT(I4SALUTE[0]);                                                 11111
     LOOP I=1,1,6 DO IF GRABONE(ZSALUTE,I-1)=0 THEN
          SW[I]=FALSE ELSE SW[I]=TRUE;
     IF PEN=6 THEN ZSALUTE=CYCLE;
     IF PEN=7 THEN ZSALUTE=T;
     IF PEN=8 THEN ZSALUTE=DT;
%I   WRITE(I4SALUTE[1],ZSALUTE);                                       11111
%%%
%%%
%%%     HERE PULSE OSK TO LET IT EXECUTE SALYUT IN DURING CLAUSE OF
%%%     ICL.  TRANSFER SALUTE INFORMATION FROM AN ILLIAC IV FILE TC THE
%%%     OUTSIDE WORLD FOR EXAMINATION.
%%%
%%%
     %
     %
     %
     IF CYCLE=TDUMP OR STOPTHISCYCLE THEN BEGIN
          RELERRNOW=(TOTALENERGYNOW-ETH)/ETH;
          NUMBEROFCYCLES=CYCLE-CYCLASTPRINT;
          IF (NUMBEROFCYCLES=0.0) THEN NUMBEROFCYCLES=1.0;
          ENGYCHECK=(RELERRNOW-RELERRBEFORE)/NUMBEROFCYCLES;
          RELERRBEFORE=RELERRNOW;
          CYCLASTPRINT=CYCLE;
          IF PEN=1 THEN Z=CYCLE;
          IF PEN=2 THEN Z=T;
          IF PEN=3 THEN Z=DT;
          IF PEN=7 THEN Z=ETH;
          IF PEN=8 THEN Z=STABLFCTR;
          IF PEN=14 THEN Z=RELERRBEFORE;
%I        WAIT(I4INOUT[JPL]);                                           11111
```

```
%I          WAIT(I4SCR[JPL]);                                                   11111
            LAST[0]=Z;
            LOOP I=0,1,IBLK DO LAST[I+1]=X[I];
            LOOP J=0,1,JMAX DO BEGIN
                 J1=J DIV 64;
                 J2=J-64*J1;
                 IF J2=PEN THEN LAST[J1+IBLK+2]=DY[J];
            END;
%I          WRITE(I4INOUT[0],Z);                                                11111
            SIMWRITE(I4DISK0[0],Z);                                             11111
%%%
%%%        INTERUPT HERE FOR DSK... WE HAVE DATA IN FILE I4INOUT TO BE
%%%        COPIED EITHER TO THE LASER STORE OR TO TAPE.  A PROGRAM IN THE
%%%        DURING CLAUSE OF ICL WILL DO THIS COPY.  ***MUST*** LOCK OUT
%%%        FURTHER WRITES BY GLYPNIR TO I4INOUT TILL THIS COPY IS COMPLETE
%%%        HENCE MUST HAVE A RESULT DESCRIPTOR FROM DSK.
%%%
%%%        THIS IS NOT YET DEFINED.  ***NOTE*** PROBABLY BETTER TO FIRST
%%%        PULSE THE COPY OF THE FILE OUT, RETURN THIS DESCRIPTOR, AND
%%%        THEN PULSE AN EDIT OF THE COPIED FILE FOR PRINTER OUTPUT.
%%%        THIS USE OF TWO TASKS IN THE DURING CLAUSE OF ICL WOULD ALLOW
%%%        MORE OVERLAP, AND WE NEED ALL WE CAN GET........
%%%
            IF RELERRNOW>MAXRELERR THEN ERROR(200);
      END;
END; %OUTPUT.
%
%
%
%
SUBROUTINE REZONE;
%          NB.  REZONE WILL READ ALL OF I4SCR AND REWRITE IT AFTER
%          THE REZONE.  IF CYCLE=TDUMP OR STOPTHISCYCLE IS TRUE THEN
%          THE FILE I4INOUT MUST BE REWRITTEN ALSO.
ERROR(100);
%
%
%
%
%           MAIN PROGRAM STARTS HERE.
%
%
SHELL
%I          OPEN(I4INOUT);                                                      11111
%I          OPEN(I4SCR);                                                        11111
%I          OPEN(I4SALUTE);                                                     11111
            INPUT; % READ Z-BLOCK,X,AND Y.  COPY HYDRO TO I4SCR.
%I          READ(I4SCR[1],LAST);                                                11111
%I          READ(I4SCR[2],NEXT);                                                11111
%I          WAIT;                                                               11111
            SIMREAD(I4DISK7[1],LAST);                                           11111
            SIMREAD(I4DISK7[2],NEXT);                                           11111
            LOOP JC=0,1,IROWSM1 DO BEGIN
                 U   [JC]=LAST[JC];
                 V   [JC]=LAST[JC+IROWS ];
                 AMX[JC]=LAST[JC+IROWS2];
                 AIX[JC]=LAST[JC+IROWS3];
                 P   [JC]=LAST[JC+IROWS4];
                 %
                 U   [JC+IROWS]=NEXT[JC];
```

```
                 V   [JC+IROWS]=NEXT[JC+IROWS  ];
                 AMX[JC+IROWS]=NEXT[JC+IROWS2];
                 AIX[JC+IROWS]=NEXT[JC+IROWS3];
                 P   [JC+IROWS]=NEXT[JC+IROWS4];
          END; %JC LOOP.
%I        READ(I4SCR[3],NEXT);                                            11111
          SIMREAD(I4DISK7[3],NEXT);                                       11111
AGAIN     CHOOSEDT;
          JPL=0;
          JPN=3;
          JBOT=0;
          LOOP JB=1,1,JSTRIPS DO BEGIN
                 PH1;
                 PH2;
                 CDT;
                 MODE=TRUE;
                 IF CYCLE=TDUMP THEN SUMTOTALENERGY;
%I               WAIT(I4SCR[JPL]); % LAST WRITE.                          11111
%I               WAIT(I4SCR[JPN]); % NEXT READ.                           11111
                 JPL=JPL+1;
                 JPN=JPN+1;
                 IF JPL=JSTRIPSP1 THEN JPL=1;
                 IF JPN=JSTRIPSP1 THEN JPN=1;
                 LOOP JC=0,1,IROWSM1 DO BEGIN
                        LAST[JC        ]=U   [JC];
                        LAST[JC+IROWS  ]=V   [JC];
                        LAST[JC+IROWS2]=AMX[JC];
                        LAST[JC+IROWS3]=AIX[JC];
                        LAST[JC+IROWS4]=P   [JC];
                        %
                        U   [JC]=U   [JC+IROWS];
                        V   [JC]=V   [JC+IROWS];
                        AMX[JC]=AMX[JC+IROWS];
                        AIX[JC]=AIX[JC+IROWS];
                        P   [JC]=P   [JC+IROWS];
                        %
                        U   [JC+IROWS]=NEXT[JC];
                        V   [JC+IROWS]=NEXT[JC+IROWS  ];
                        AMX[JC+IROWS]=NEXT[JC+IROWS2];
                        AIX[JC+IROWS]=NEXT[JC+IROWS3];
                        P   [JC+IROWS]=NEXT[JC+IROWS4];
                 END; %JC LOOP.
                 IF CYCLE=TDUMP OR STOPTHISCYCLE THEN
%I                      WRITE(I4INOUT[JPL],LAST);                         11111
                        SIMWRITE(I4DISK0[JPL],LAST);                      11111
%I               WRITE(I4SCR[JPL],LAST);                                  11111
%I               READ(I4SCR[JPN],NEXT);                                   11111
                 SIMWRITE(I4DISK7[JPL],LAST);                             11111
                 SIMREAD(I4DISK7[JPN],NEXT);                              11111
                 JBOT=JBOT+JMAXPERSTRIP;
          END; %JB LOOP.
          OUTPUT;
          IF REZONETOP OR REZONERIGHT OR REZONEBOTTOM THEN REZONE;
          IF NOT STOPTHISCYCLE THEN GO TO AGAIN;
%
STOP FIN END.
```

APPENDIX XII

CODE REQUIRED TO TRANSFER A B6500 CLAM OUTPUT FILE TO AN ILLIAC IV INPUT FILE

```
$LIST SINGLE
FILE  4=CLAM/OUT,UNIT=DISK,RECORD=1000
FILE  1=CLAM/CUTG,UNIT=DISK,SAVE=1,AREA=60,RECORD=540
C
C        PROGRAM I4CLAM FOR SHELL/OF/THE/FUTURE.
C        THIS DECK WILL CONVERT ANY SIZE CLAM WRITTEN 1000 WDS/RECORD.
C
       REAL   Z(150),U(200),V(200),AMX(200),AIX(200),P(200),X(1000),
      C   Y(1500),ZZ(16000)
       EQUIVALENCE (IMAX,Z(33)), (JMAX,Z(35)), (KMAXA,Z(38))
       INTEGER STRIPS
       LOGICAL FIRST
       DATA FIRST/.TRUE./
C
C
       STRIPS=5
       IROWSH=50
       KLH=5*64*IROWSH
C
       READ(4) WS,WSA
       READ(4) Z
C
       JROWS=JMAX-1
       JPS=JROWS/STRIPS
       IF(JPS*STRIPS.EQ.JROWS) GO TO 6
       WRITE(6,5)
5      FORMAT(15H BAD CLAM DATA.)
       STOP
C
6      K1=1
10     KL=KMAXA
       IF(KL.GT.K1+199) KL=K1+199
       READ (4) DMY
       K1=KL+1
       IF(KL.LT.KMAXA) GO TO 10
C
       READ(4) X0,(X(I),DMY,I=1,IMAX)
       READ(4) Y0,(Y(J),J=1,JMAX)
C
       DO 20 K=1,KLH
20     ZZ(K)=0
       ZZ(1)=Z(1)
       ZZ(2)=Z(2)
       ZZ(3)=Z(84)
       ZZ(4)=Z(3)
       ZZ(5)=Z(26)
       ZZ(6)=IMAX
       ZZ(7)=JROWS
       ZZ(8)=Z(13)
       ZZ(9)=Z(139)
       ZZ(10)=Z(108)
       ZZ(11)=Z(75)
       ZZ(12)=0
       ZZ(13)=0
       ZZ(14)=1
       ZZ(15)=0
       ZZ(16)=1
       ZZ(17)=1.
       ZZ(18)=JPS
C
```

```
      DO 40 I=1,IMAX
40    ZZ(I+65)=X(I)
C
      IBLKP1=(IMAX+65)/64
      IWDS1=64*IBLKP1
      IROWS=IBLKP1*JPS
      IWDS=64*IROWS
      IWDSY=IWDS1+64
C
      KU=0
      KV=KU+IWDS
      KM=KV+IWDS
      KI=KM+IWDS
      KP=KI+IWDS
C
      DO 60 J=1,JMAX
60    ZZ(J+IWDSY)=Y(J)
C
      CALL WRTI4D(ZZ,KLH)
C
      REWIND 4
      READ(4) DMY
      READ(4) DMY
      DO 50 K=1,KLH
50    ZZ(K)=0
C
      KN=1
      K=200
      DO 100 J=2,JMAX
      KL=KN
      DO 95 I=1,IMAX
70    IF(K.LT.200) GO TO 80
      READ(4) (U(KK),V(KK),AMX(KK),AIX(KK),P(KK),KK=1,200)
      K=K-200
      IF (FIRST) K=IMAX+1
      FIRST=.FALSE.
      GO TO 70
C
80    K=K+1
      KN=KN+1
      ZZ(KN+KU)=U(K)
      ZZ(KN+KV)=V(K)
      ZZ(KN+KM)=AMX(K)
      ZZ(KN+KI)=AIX(K)
      ZZ(KN+KP)=P(K)
95    CONTINUE
      KN=KL+IWDS1
      IF(((J-1)/JPS)*JPS.NE.J-1) GO TO 100
      CALL WRTI4D(ZZ,KLH)
      DO 96 I=1,KLH
96    ZZ(I)=0
      KN=1
100   CONTINUE
      LOCK1
      STOP
      END
      SUBROUTINE WRTI4D(Z,N)
      DIMENSION A(540),Z(N)
      WRITE(6,99)
99    FORMAT(1H1)
```

```
      LL=0
      NN=N
5     K=NN
      IF(K.GT.256) K=256
      DO 11 I=1,K
      L=LL+I
      IF(Z(L).EQ.0) GO TO 10
      WRITE(6,98) L,Z(L),Z(L)
98    FORMAT(I10,F20.8,E20.8)
      CALL B5I4F(Z(L),A(2*I-1),A(2*I))
      GO TO 11
10    A(2*I-1)=0.
      A(2*I)=0.
11    CONTINUE
      WRITE(1) (A(I),I=1,540)
      NN=NN-K
      IF(NN.EQ.0) RETURN
      LL=LL+256
      GO TO 5
      END
      SUBROUTINE B5I4F(A,B,C)
      EQUIVALENCE (XX,II)
C
      XE=CONCAT(0,A,6,45,7)
      IF(XE.EQ.0) GO TO 10
      CALL BI4F(A,B,C)
      RETURN
C
10    XX=A
      X=II
      X=X+0.1
      X=X-0.1
      CALL BI4F(X,B,C)
      RETURN
      END
      SUBROUTINE BI4F(A,B,C)
      EQUIVALENCE (XE,IE8)
      DATA IBIAS/040000/
      IF(A.NE.0) GO TO 10
      B=0
      C=0
      RETURN
C
10    XS=CONCAT(0,A,0,45,1)
      XE=CONCAT(0,A,5,44,6)
      IF(XS.NE.0) IE8=-IE8
      IE8=-IE8
      B9=CONCAT(0,A,0,38,1)
      B10=CONCAT(0,A,0,37,1)
      IE8E=2
      IF(B10.NE.0) IE8E=1
      IF(B9.NE.0) IE8E=0
      IE=3*IE8+IE8E
      IE=IBIAS-IE+39
      BB=CONCAT(0,A,31,46,1)
      BB=CONCAT(BB,IE,30,14,15)
      B=CONCAT(BB,A,15,38-IE8E,16)
      C=CONCAT(0,A,31,22-IE8E,23-IE8E)
      RETURN
      END
```

```
SUBROUTINE BI4I(A,B,C)
BB=CONCAT(0,A,31,46,1)
B=CONCAT(BB,A,6,38,7)
C=CONCAT(0,A,31,31,32)
RETURN
END
```

APPENDIX XIII

CODE REQUIRED TO EDIT AN OUTPUT DUMP FROM SHELL/OF/THE/FUTURE

```
$LIST SINGLE
FILE  4=SHELL/I4CLAM/OUT60BY15,UNIT=DISK,RECORD=540
C
C         EDIT FOR SHELL/OF/THE/FUTURE.
C
      COMMON Z(16000)
      DIMENSION PR(4)
      DIMENSION U(1000),V(1000),AMX(1000),AIX(1000),P(1000),
     C   X(1000),DX(1000),TAU(1000),Y(1500),DY(1500)
C
C
      IWDST=5*64*50
C
205   CALL REDI4D(Z,IWDST)
      CALL PRINTZ
C
      PROB=Z(1)
      CYCLE=Z(2)
      T=Z(3)
      DT=Z(4)
      IMAX=Z(6)
      JMAX=Z(7)
      ETH=Z(8)
      RELERR=Z(15)
      JPS=Z(18)
      STRIPS=JMAX /JPS
      IBLKP1=(IMAX+65)/64
      IWDS1=64*IBLKP1
      IROWS=IBLKP1*JPS
      IWDS=64*IROWS
      IWDSY=IWDS1+64
      KU=0
      KV=KU+IWDS
      KM=KV+IWDS
      KI=KM+IWDS
      KP=KI+IWDS
C
      DO 10 I=1,IMAX
10    X(I)=Z(I+65)
C
      WSA=0
      DO 15 I=1,IMAX
      DX(I)=X(I)-WSA
15    WSA=X(I)
C
      DO 20 J=1,JMAX
20    Y(J)=Z(J+IWDSY)
C
      WSA=0
      DO 25 J=1,JMAX
      DY(J)=Y(J)-WSA
25    WSA=Y(J)
C
      WSA=0
      DO 30 I=1,IMAX
      WSB=X(I)**2
      TAU(I)=3.14159265358979*(WSB-WSA)
30    WSA=WSB
C
      KK=0
```

```
      DO 600 JB=1,STRIPS
      CALL REDI4J(Z,IWDST)
      KL=0
      DO 610 J=1,JPS
      DO 620 I=1,IMAX
      KK=KK+1
      K=KL+I+1
      U(KK)=Z(K+KU)
      V(KK)=Z(K+KV)
      AMX(KK)=Z(K+KM)
      AIX(KK)=Z(K+KI)
      P(KK)=Z(K+KP)
620   CONTINUE
      KL=KL+IWDS1
610   CONTINUE
600   CONTINUE
C
C        SHORT PRINT
C
      WRITE(6,470) CYCLE
      DO 200 I=1,4
200   PR(I)=0
      K=0
      DO 221 J=1,JMAX
      DO 220 I=1,IMAX
      K=K+1
      IF(AMX(K))300,220,210
210   WSB=0.5*(U(K)**2+V(K)**2)
      PR(1)=PR(1)+AMX(K)*AIX(K)
      PR(2)=PR(2)+WSB*AMX(K)
      PR(4)=PR(4)+AMX(K)
220   CONTINUE
221   CONTINUE
      PR(3)= PR(1)+PR(2)
      WRITE(6,360) PROB,CYCLE,T,DT
      WRITE(6,370) (PR(I),I=1,4)
      WRITE(6,380) ETH,RELERR
C
C        LONG PRINT CODE.
C
      WRITE(6,360) PROB,CYCLE,T,DT
      DO 280 I=1,IMAX
      WRITE(6,420) I,X(I),DX(I)
      K=IMAX*JMAX+I
      J=JMAX+1
      DO 270 L=1,JMAX
      J=J-1
      K=K-IMAX
      IF(AMX(K).EQ.0.) GO TO 270
      RHO=AMX(K)/(TAU(I)*DY(J))
      WRITE(6,430) J,U(K),V(K),P(K),DMY,AMX(K),AIX(K),RHO,Y(J)
270   CONTINUE
280   WRITE(6,500)
      GO TO 205
C
C        NEGATIVE MASS.
C
300   WRITE(6,490)
      STOP
C
```

```
C        FORMATS.
C
360   FORMAT(8H1PROBLEM6X5HCYCLE9X4HTIME13X2HDT/F9.4,F11.0,1P2X2E16.7)
370   FORMAT(////17X2HAI16X2HAK14X5HAI+AK15X2HAM/1P7X4E18.7)
380   FORMAT(///16X3HTHE12X9HREL ERROR/1P7X2E18.7////)
420   FORMAT(4H I =I3,6X6HX(I) =F12.3,6X7HDX(I) =F12.3//3H  J8X1HU
     X13X1HV13X1HP14X4X11X3HAMX11X3HAIX12X3HRHO11X1HY/)
430   FORMAT(1P1XI3,1X8E14.6)
470   FORMAT(///21H TAPE 4 OUMP ON CYCLEF6.0////)
490   FORMAT(27H1NEGATIVE MASS ENCOUNTERED. )
500   FORMAT(/////)
      END
      SUBROUTINE PRINTZ
      COMMON Z(64)
C
      WRITE(6,99)
99    FORMAT(1H1)
      DO 20 I=1,32
      II=I+32
20    WRITE(6,10) I,Z(I),Z(I),II,Z(II),Z(II)
10    FORMAT(1XI10,1PE20.6,0P1XF20.6,15X,I10,1PE20.6,0P1XF20.6)
      RETURN
      END
      SUBROUTINE REDI4D(Z,N)
      REAL Z(N),ZZZ(540)
      LOGICAL FIRST
      DATA FIRST/.TRUE./
C
      IF(FIRST) REWIND 4
      FIRST=.FALSE.
      N1=0
100   READ(4,END=200) (ZZZ(I),I=1,540)
      DO 11 K=1,256
      I=N1+K
      CALL I4BF(ZZZ(2*K-1),ZZZ(2*K),Z(I))
10    IF(I.GE.N) RETURN
11    CONTINUE
      N1=N1+256
      GO TO 100
C
200   WRITE(6,201)
201   FORMAT(24H1END OF FILE ON I4 DISK.)
      STOP
      END
      SUBROUTINE I4BF(A,B,C)
      INTEGERO100
      DATA IBIAS,O100/040000,0100/
      EQUIVALENCE(XE,IE)
C
C
      IF(A.NE.0.0.OR.B.NE.0.0) GO TO 5
      C=0
      RETURN
C
5     XE=CONCAT(0,A,14,30,15)
      IE=IBIAS+39-IE
C      IE8 IS POWER OF 8
10    IE8=IE/3
C      IE8E IS RT SHIFT OF MANTISSA AT IE8
      IE8E=IE-3*IE8
```

```
      IF(IE8E.GE.0) GO TO 20
      IE8E=IE8E+3
      IE8=IE8-1
20    IE8=-IE8
      IF(IE8.LT.0) IE8=0100-IE8
C      PICK UP SIGN
      OUT=CONCAT(0,A,46,31,1)
C      PICK UP EXPONENT
      OUT=CONCAT(OUT,IE8,45,6,7)
C      PICK UP TOP OF MANTISSA
      OUT=CONCAT(OUT,A,38-IE8E,15,16)
C      PICK UP BOTTOM OF MANTISSA
      C=CONCAT(OUT,B,22-IE8E,31,23-IE8E)
      RETURN
      END
      SUBROUTINE I4BI(A,B,C)
      IF(A.NE.0.0.OR.B.NE.0.0) GO TO 5
      C=0
      RETURN
C
5     CC=CONCAT(0,A,46,31,1)
      CC=CONCAT(CC,A,38,6,7)
      C=CONCAT(CC,B,31,31,32)
      RETURN
      END
```

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)

Air Force Weapons Laboratory (SYT)
Kirtland Air Force Base, New Mexico 87117

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

3. REPORT TITLE

SHELL FOR THE ILLIAC IV

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

1 October 1970-31 August 1971

5. AUTHOR(S) (First name, middle initial, last name)

Major William A. Whitaker, USAF; Captain Richard E. Durrett, USAF;
Captain Reginald W. Clemens, USAF

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March 1972	116	None

8a. CONTRACT OR GRANT NO.

b. PROJECT NO. ARPA Order No. 17446T

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

AFWL-TR-72-33

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

AFWL (SYT)
Kirtland AFB, NM 87117

13. ABSTRACT

(Distribution Limitation Statement A)

A one-dimensional Lagrangian hydrodynamics computer code (SAP) and a two-dimensional Eulerian hydrodynamics computer code (SHELL) have been successfully written in the GLYPNIR language for the ILLIAC IV. Timing simulations suggest a speed 50 times that of a CDC 6600 for the GLYPNIR SHELL code.

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
ILLIAC IV						
Hydrodynamic computer code						
SHELL code						
Computer programs						
Two-dimensional hydrodynamics						
Hydrodynamics						

UNCLASSIFIED
Security Classification